ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Desarrollo de una aplicación Android para administración y gestión de la información de registro de servicios en una plataforma SOA.

AUTOR: Luis José Úbeda Aguilar

TITULACIÓN: Telemática

TUTOR (o Director en su caso): Rubén De Diego Martínez

DEPARTAMENTO: DIATEL (Departamento de Ingeniería y Arquitecturas Telemáticas)

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Antonio Pedrero González

VOCAL: Rubén de Diego Martínez

SECRETARIO: Gregorio Rubio Cifuentes

Fecha de lectura:

Calificación:

El Secretario,

Agradecimientos

A mi familia, en especial a mis padres, César y Maria, mi hermana Leticia, mi abuela Amelia y mi primo Kevin.

A mi mejor amigo Carlos, por haber estado presente siempre desde que tengo uso de razón, a sus padres José y Rosa, y su hermano Alberto, que son mi segunda familia.

A mis amigos, en especial a Francisco, Daniel, Juan Miguel, Raquel, Ana Belén, Cristina, Patricia, Javier y Aitor, por todos estos años de amistad. ¡Y que duren muchos más!

A mis compañeros de universidad Carlos y Jose María, de los que me llevo dos amigos para siempre.

A Rubén por haber confiado en mí para realizar este proyecto, y por sus años de enseñanza durante toda la carrera.

A Lourdes y Juan por haberme abierto las puertas del CITSEM y haberme ofrecido la oportunidad de colaborar con ellos.

A todos los miembros del grupo de investigación GRyS, en especial a Jesús, Alexandra y Néstor, por toda la ayuda ofrecida durante la realización del proyecto.

Resumen

Este Proyecto Fin de Grado está enmarcado dentro de las actividades del GRyS (Grupo de Redes y Servicios de Próxima Generación) con las *Smart Grids*.

En la investigación actual sobre *Smart Grids* se pretenden alcanzar los siguientes objetivos:

- Integrar fuentes de energías renovables de manera efectiva.
- Aumentar la eficiencia en la gestión de la demanda y suministro de forma dinámica.
- Reducir las emisiones de CO_2 dando prioridad a fuentes de energía verdes.
- Concienciar del consumo de energía mediante la monitorización de dispositivos y servicios.
- Estimular el desarrollo de un mercado vanguardista de tecnologías energéticamente eficientes con nuevos modelos de negocio.

Dentro del contexto de las *Smart Grids*, el interés del GRyS se extiende básicamente a la creación de *middlewares* semánticos y tecnologías afines, como las ontologías de servicios y las bases de datos semánticas.

El objetivo de este Proyecto Fin de Grado ha sido diseñar y desarrollar una aplicación para dispositivos con sistema operativo Android, que implementa una interfaz gráfica y los métodos necesarios para obtener y representar información de registro de servicios de una plataforma SOA (*Service-Oriented Architecture*). La aplicación permite:

- Representar información relativa a los servicios y dispositivos registrados en una *Smart Grid*.
- Guardar, cargar y compartir por correo electrónico ficheros HTML con la información anterior.
- Representar en un mapa la ubicación de los dispositivos.
- Representar medidas (voltaje, temperatura, etc.) en tiempo real.
- Aplicar filtros por identificador de dispositivo, modelo o fabricante.
- Realizar consultas SPARQL a bases de datos semánticas.
- Guardar y cagar consultas SPARQL en ficheros de texto almacenados en la tarjeta SD.

La aplicación, desarrollada en Java, es de código libre y hace uso de tecnologías estándar y abiertas como HTML, XML, SPARQL y servicios RESTful. Se ha tenido ocasión de probarla con la infraestructura del proyecto europeo e-Gotham (*Sustainable-Smart Grid Open System for the Aggregated Control, Monitoring and Management of Energy*), en el que participan 17 socios de 5 países: España, Italia, Estonia, Finlandia y Noruega.

En esta memoria se detalla el estudio realizado sobre el Estado del arte y las tecnologías utilizadas en el desarrollo del proyecto, la implementación, diseño y arquitectura de la aplicación, así como las pruebas realizadas y los resultados obtenidos.

Abstract

This Final Degree Project is framed within the activities of the GRyS (Grupo de Redes y Servicios de Próxima Generación) with the Smart Grids.

Current research on Smart Grids aims to achieve the following objectives:

- To effectively integrate renewable energy sources.
- To increase management efficiency by dynamically matching demand and supply.
- To reduce carbon emissions by giving priority to green energy sources.
- To raise energy consumption awareness by monitoring products and services.
- To stimulate the development of a leading-edge market for energy-efficient technologies with new business models.

Within the context of the Smart Grids, the interest of the GRyS basically extends to the creation of semantic middleware and related technologies, such as service ontologies and semantic data bases.

The objective of this Final Degree Project has been to design and develop an application for devices with Android operating system, which implements a graphical interface and methods to obtain and represent services registry information in a Service-Oriented Architecture (SOA) platform. The application allows users to:

- Represent information related to services and devices registered in a Smart Grid.
- Save, load and share HTML files with the above information by email.
- Represent the location of devices on a map.
- Represent measures (voltage, temperature, etc.) in real time.
- Apply filters by device id, model or manufacturer.
- SPARQL query semantic database.
- Save and load SPARQL queries in text files stored on the SD card.

The application, developed in Java, is open source and uses open standards such as HTML, XML, SPARQL and RESTful services technologies. It has been tested in a real environment using the e-Gotham European project infrastructure (Sustainable-Smart Grid Open System for the Aggregated Control, Monitoring and Management of Energy), which is participated by 17 partners from 5 countries: Spain, Italy, Estonia, Finland and Norway.

This report details the study on the State of the art and the technologies used in the development of the project, implementation, design and architecture of the application, as well as the tests performed and the results obtained.

Índice

Índice de figuras

Índice de figuras

Acrónimos

CIM: Common Information Model.

DHCP: Dynamic Host Configuration Protocol.

DNS: Domain Name System.

DTD: Document Type Definition.

FOAF: Friend of a Friend.

EA: Enterprise Architecture.

EAI: Enterprise Application Integration.

EMS: Energy Management System.

ETP Smart Grids: European Technology Platform for Electricity Networks of the Future.

HAN: Home Area Network.

HTTP: Hypertext Transfer Protocol.

IEC: International Electrotechnical Commission.

IETF: Internet Engineering Task Force.

IP: Internet Protocol.

IOT: Internet of Things.

IT: Information Technology.

JVM: Java Virtual Machine.

LUS: Look Up Service.

NIST: National Institute of Standards and Technology.

OASIS: Organization for the Advancement of Structured Information Standards.

OWL: Web Ontology Language.

RDF: Resource Description Framework.

SAX: Simple API for XML

SCADA: Supervisory Control and Data Acquisition.

SLP: Service Location Protocol.

SOA: Service Oriented Architecture.

SOAP: Simple Object Access Protocol.

SPARQL: SPARQL and RDF Query Language.

TCP: Transmission Control Protocol.

UDDI: Universal Description, Discovery, and Integration.

UPnP: Universal Plug and Play.

URI: Universal Resource Identifier.

URL: Universa Resource Link.

WSDL: Web Services Description Language.

XML: eXtensible Markup Language.

W3C: World Wide Web Consortium.

WSMF: Web Service Modeling Framework.

WSML: Web Service Modeling Language.

WSMO: Web Service Modeling Ontology.

1 Introducción

El concepto de *Smart Grids* fue desarrollado en 2006 por la *European Technology Platform for Electricity Networks of the Future* (ETP Smart Grids) [1] y se refiere a una red eléctrica que puede integrar de forma inteligente las acciones de todos los usuarios conectados a ella – generadores, consumidores y aquellos que hacen ambas cosas – con el fin de administrar suministro eléctrico de forma sostenible, económica y segura. Una red inteligente cuenta con productos y servicios innovadores, junto con la monitorización, control, comunicación y tecnologías de resiliencia (capacidad de un sistema de soportar y recuperarse ante desastres y perturbaciones) con el fin de:

- Facilitar la conexión y el funcionamiento de generadores de todos los tamaños y tecnologías.
- Permitir que los consumidores jueguen un papel en la optimización del funcionamiento del sistema.
- Proveer a los consumidores con más información y mejores opciones a la hora de elegir su proveedor de energía.
- Reducir significantemente el impacto medioambiental de todo el sistema de suministro de electricidad.
- Mantener y mejorar los altos niveles de confiabilidad del sistema, la calidad y seguridad del suministro existente.
- Fomentar el desarrollo de un mercado europeo integrado.

Dentro de este nuevo paradigma, el interés de este Proyecto de Fin de Grado se extiende al ámbito del registro y monitorización de servicios y dispositivos en una *Smart Grid*. Para ello, se ha desarrollado una aplicación para dispositivos con sistema operativo Android, que implementa una interfaz gráfica y los métodos necesarios para obtener y representar información de registro de servicios de una plataforma SOA (*Service-Oriented Architecture*). La aplicación permite:

- Representar información relativa a los servicios y dispositivos registrados en una *Smart Grid*.
- Guardar, cargar y compartir por correo electrónico ficheros HTML con la información anterior.
- Representar en un mapa la ubicación de los dispositivos.
- Representar medidas (voltaje, temperatura, etc.) en tiempo real.
- Aplicar filtros por identificador de dispositivo, modelo o fabricante.
- Realizar consultas SPARQL a bases de datos semánticas.
- Guardar y cagar consultas SPARQL en ficheros de texto almacenados en la tarjeta SD.

La aplicación, desarrollada en Java, es de código libre y hace uso de tecnologías estándar y abiertas como HTML, XML o SPARQL. Se probará usando la infraestructura de un proyecto europeo de investigación, haciendo uso de las interfaces RESTful para acceder a los diferentes servicios que dispone la plataforma.

2 Las Smart Grids o redes inteligentes

2.1 Introducción

Según la definición de la Red Eléctrica Española (REE) [2]:

"Una red inteligente es aquella que puede integrar de forma eficiente el comportamiento y las acciones de todos los usuarios conectados a ella, de tal forma que se asegure un sistema energético sostenible y eficiente, con bajas pérdidas y altos niveles de calidad y seguridad de suministro."

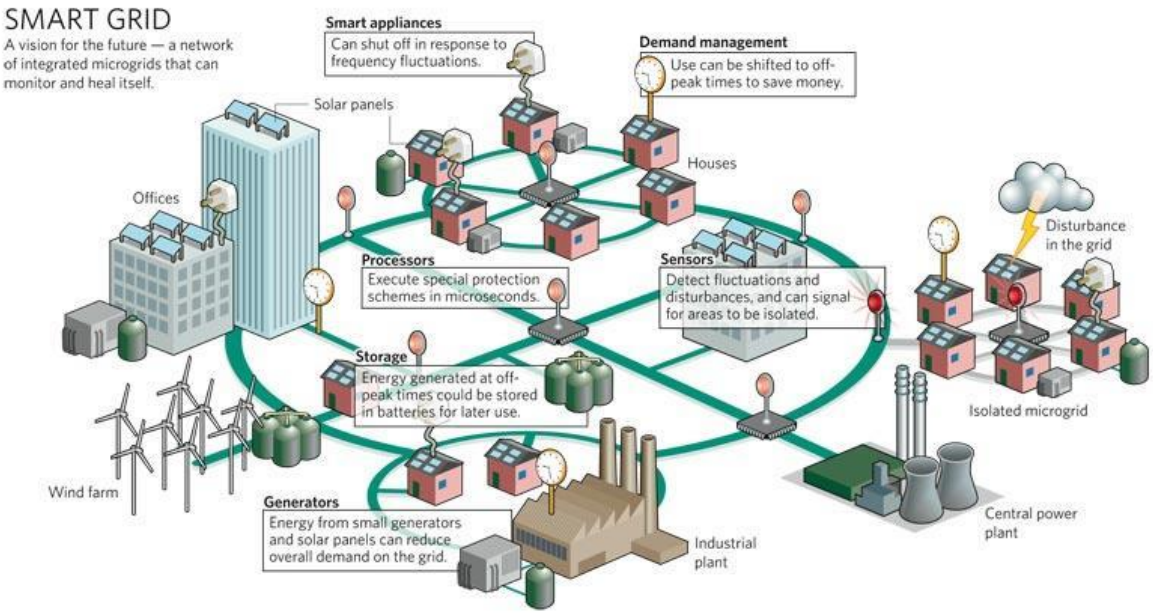

Figura 1: Ecosistema de una Smart Grid.[1]

Las redes inteligentes o *Smart Grids* definen la mejora de la infraestructura de los segmentos que componen el sistema de suministro de energía [3]:

- Generación de energía.
- Transmisión.
- Distribución.
- Consumo.

Una *Smart Grid* proporciona la automatización necesaria para administrar los recursos de energía mediante la mejora de su explotación, minimizando los residuos e informando en tiempo real tanto a proveedores como a consumidores. Se requiere para ello una infraestructura moderna, inexistente hasta el momento, que maximice la entrada y distribución de energía y que su vez sea económica de operar y mantener. Estos cambios harán que se ahorre en costes y mejorará la eficiencia de la red.

[1] Fuente: http://www.smartgrid2030.com/

A continuación se citan algunas de las características de una red inteligente:

- Monitorización remota y control de la producción y consumo de energía.
- Medición precisa utilizando tecnología digital.
- Reducción de costes de electricidad debido a un consumo más preciso y sensible.
- Mejora en la toma de decisiones del usuario final sobre su consumo de energía.
- Gestión más eficaz de la red por parte de los proveedores.
- Relación entre proveedores y consumidores de energía con más información y cooperación.

Los defensores de la red inteligente creen que va a abrir nuevos mercados para los productores a pequeña y gran escala de energías alternativas como la solar, la eólica y los biocombustibles mediante la descentralización de la generación de energía. A largo plazo, la transparencia ofrecida por la red eléctrica inteligente abre nuevos mercados y modelos de negocio. La exitosa implementación de la infraestructura se enfrenta a dos obstáculos: los estándares y la seguridad.

Las perspectivas mundiales en el mercado de las *Smart Grids* son favorables. Dicho mercado cuenta con un crecimiento medio anual de un 26% hasta 2017 [4]. El volumen de ese mercado global se incrementará desde los $23.970.000.000 en 2010 hasta los $125.150.000.000 en 2017.

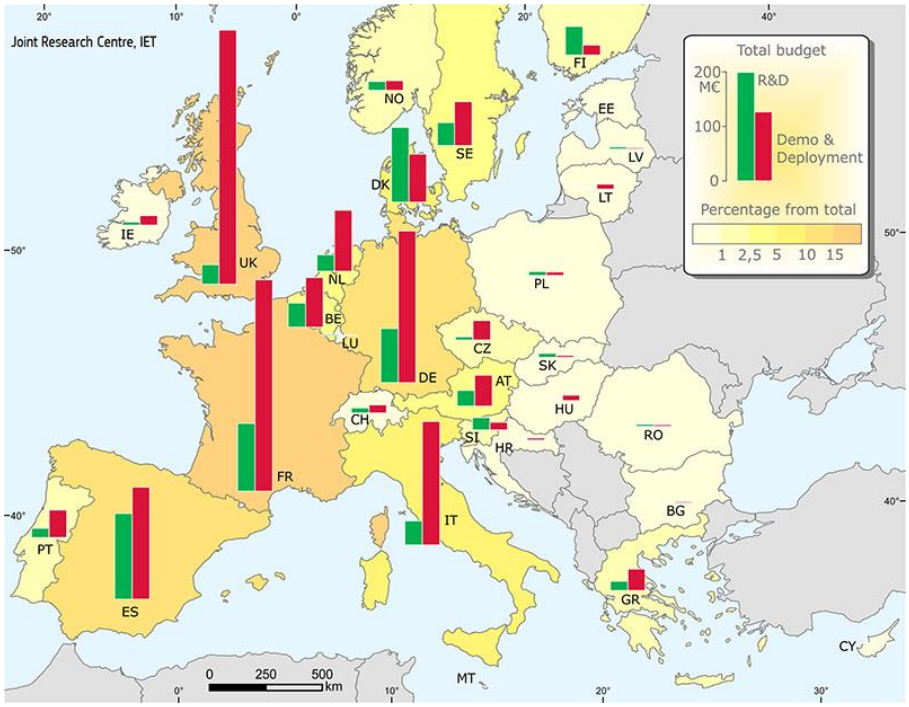

Figura 2: Inversión europea en I+D y despliegue de Smart Grids.[2]

Respecto al ámbito de desarrollo mundial, actualmente las mayores inversiones se efectúan en Estados Unidos y en la Unión Europea (aproximadamente un 70% del total). En cambio, en 2017, se verá disminuido el peso de estas regiones hasta un 12% del total, mientras que países en vías de desarrollo, sobre todo en Asia y Latinoamérica, pasarán a ser el mayor mercado de esta tecnología.

[2] Fuente: http://ses.jrc.ec.europa.eu/

2.2 Proyectos de investigación relacionados con la Smart Grid

2.2.1 e-Gotham

e-GOTHAM (Sustainable-Smart Grid Open System for the Aggregated Control, Monitoring and Management of Energy) define una solución completa para *microgrids* en sectores residenciales, terciarios e industriales. Sus principales objetivos son [5]:

- Integrar fuentes de energías renovables de manera efectiva.
- Aumentar la eficiencia en la gestión de la demanda y suministro de forma dinámica.
- Reducir las emisiones de CO_2 dando prioridad a fuentes de energía verdes.
- Concienciar del consumo de energía mediante la monitorización de dispositivos y servicios.
- Estimular el desarrollo de un mercado vanguardista de tecnologías energéticamente eficientes con nuevos modelos de negocio.

e-GOTHAM es un proyecto de cooperación europea liderado por Inabensa, en el que participan 17 entidades europeas. En España, está siendo cofinanciado por Artemis JU y el Ministerio de industria, Energía y Turismo (Ref: Art-010000-2012-3), en el marco del programa europeo Artemis, que define e implementa la agenda de investigación para el desarrollo de tecnologías clave en el campo de sistemas informáticos integrados.

Figura 3: Consorcio e-Gotham.[3]

[3] Fuente: http://www.e-gotham.eu/ y elaboración propia

2.2.2 I3RES

I3RES tiene como objetivo integrar las energías renovables en la red de distribución con la incorporación de inteligencia en tres niveles diferentes: en la integración de fuentes de energía renovables – *Renewable Energy Sources* (RES) – y el desarrollo de mecanismos de control y de gestión que reduzcan el impacto de su intermitencia; en la facilitación de la participación de todos los actores en el mercado de la electricidad; y en el funcionamiento global de la red [6]. El consorcio está formado por ocho empresas de diferentes países europeos.

La meta principal de I3RES es desarrollar una herramienta de gestión para la red de distribución apoyada por:

- Un sistema de control que integre la información de los sistemas ya instalados (como por ejemplo SCADA, EMS y medidores inteligentes).
- Algoritmos de predicción de producción de energía y de gestión de red que ayuden a la empresa distribuidora en la gestión de la producción RES a gran escala.
- Minería de datos – *Data mining* – e inteligencia artificial para analizar la demanda de energía de los consumidores y la producción de la red de distribución.

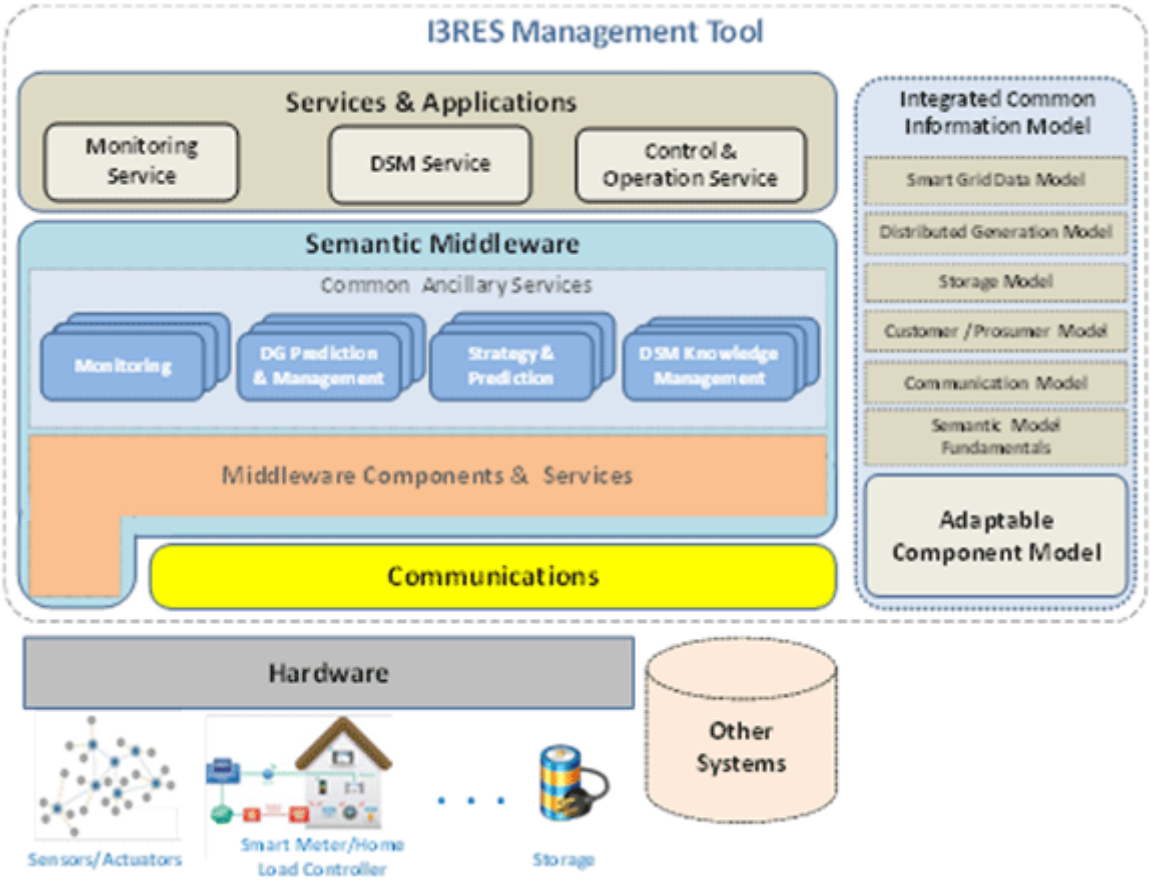

Figura 4: Herramienta de gestión I3RES.[4]

[4] Fuente: http://www.i3res.eu/v1/images/i3resconcept.png

2.2.3 Arrowhead

Arrowhead es uno de los proyectos de investigación del ámbito de las *Smart Grids* más ambiciosos. Cuenta con 77 participantes de 15 países europeos y una duración de 4 años (2013-2017). El proyecto gira alrededor de la idea de Automatización Cooperativa – *Cooperative Automation* – de las interacciones dinámicas entre productores de energía, consumidores, máquinas y sistemas [7]. La base fundamental es tecnología desarrollada en torno al Internet de las Cosas y las arquitecturas orientadas a servicios.

Los principales objetivos del proyecto son:

- Proporcionar un *framework* para el desarrollo de servicios para *Smart Grids* adaptado en términos de rendimiento y funcionalidad.
- Ofrecer soluciones para la integración con los sistemas actuales.
- Implementar y evaluar la automatización cooperativa a través de diversas aplicaciones: vehículos eléctricos, edificios inteligentes, ciudades e infraestructuras, producción industrial, producción de energía y mercados virtuales de energía.
- Mostrar las innovaciones disponibles a través de nuevos servicios y el camino en los trabajos de normalización.

La estrategia adoptada en el proyecto tiene 4 grandes dimensiones:

- Una estrategia de innovación basada en el análisis de las deficiencias del negocio, combinada con una estrategia enfocada a las necesidades del usuario final así como las tecnologías utilizadas a largo plazo.
- Aplicaciones piloto en el caso de demostraciones en entornos reales.
- Un *framework* que permita autorización colaborativa y la innovación tecnológica de vulnerabilidades críticas.
- Una metodología de coordinación.

El proyecto Arrowhead está cofinanciado por Artemis JU y el ministerio de Industria, Energía y Turismo (Ref: ART-010000-2013-3).

Figura 5: Logotipo del proyecto Arrowhead [5].

[5] Fuente: http://www.arrowhead.eu/

2.2.4 e-Highway2050

En respuesta al call ENERGY.2012.7.2.1 [8], el objetivo primordial del e-Highway2050 es desarrollar una metodología de planificación para proporcionar una primera versión de un plan de expansión modular para la Red de Transmisión Pan-Europea – *Pan-European Transmission Network* – desde el año 2020 hasta el 2050, en consonancia con los fundamentos de la política energética europea [9].

Se trata así pues, de un proyecto a largo plazo que demuestra el interés europeo por transformar las redes de distribución eléctricas convencionales en *Smart Grids*, pensando en ellas como las redes del futuro. El primer paso será implementar un conjunto de escenarios futuros del sistema de potencia: estos incluyen los requisitos para el respaldo y balanceo de carga de los generadores y el almacenamiento de la electricidad. Los resultados se presentaran y debatirán con los representantes de las partes interesadas en Europa para abordar las principales barreras potenciales y las opciones de arquitecturas propuestas. Por último, los aspectos de normalización serán analizados por las autoridades de regulación europeas.

El plan de trabajo define los procesos y el flujo de trabajo para el establecimiento del Plan de Desarrollo Modular – *Modular Development Plan* – dividido en una serie de paquetes de trabajo individuales tal como se muestra en la figura a continuación:

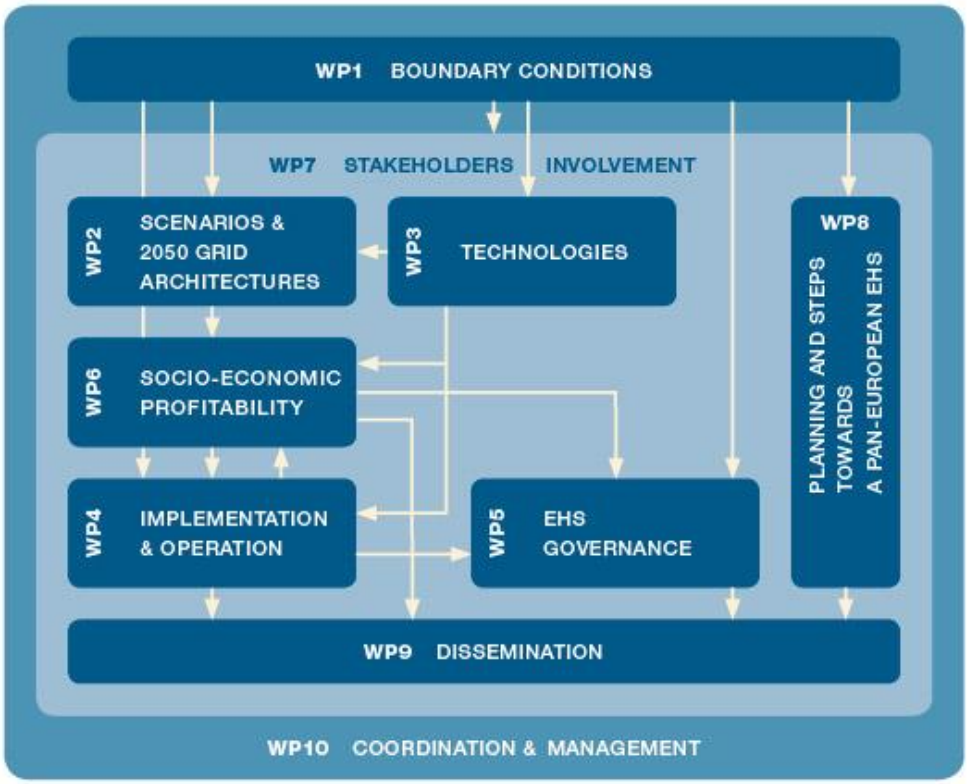

Figura 6: Plan de trabajo del proyecto e-Highway2050.[6]

[6] Fuente: http://www.e-highway2050.eu/

3 Plataforma SOA

3.1 Introducción

En las arquitecturas usadas para *Smart Grid* es muy habitual la utilización de plataformas SOA (*Service Oriented Architecture*). SOA es un paradigma de arquitectura para diseñar y desarrollar sistemas distribuidos de forma fácilmente escalable [10]. Es independiente de cualquier proveedor, producto tecnología.

Un servicio es una unidad autónoma de funcionalidad. Cada servicio implementa al menos una acción, como obtener el extracto de una cuenta bancaria o modificar o modificar la reserva de un billete de avión.

Los servicios pueden ser combinados con otras aplicaciones de *software* para proporcionar la funcionalidad completa de una aplicación más compleja. SOA facilita la cooperación entre los equipos conectados a una red. Cada equipo puede ejecutar un número arbitrario de servicios, y cada servicio es construido de una manera que garantiza que pueda intercambiar información con cualquier otro servicio de la red sin necesidad de intervención humana ni realizar cambios en el propio programa.

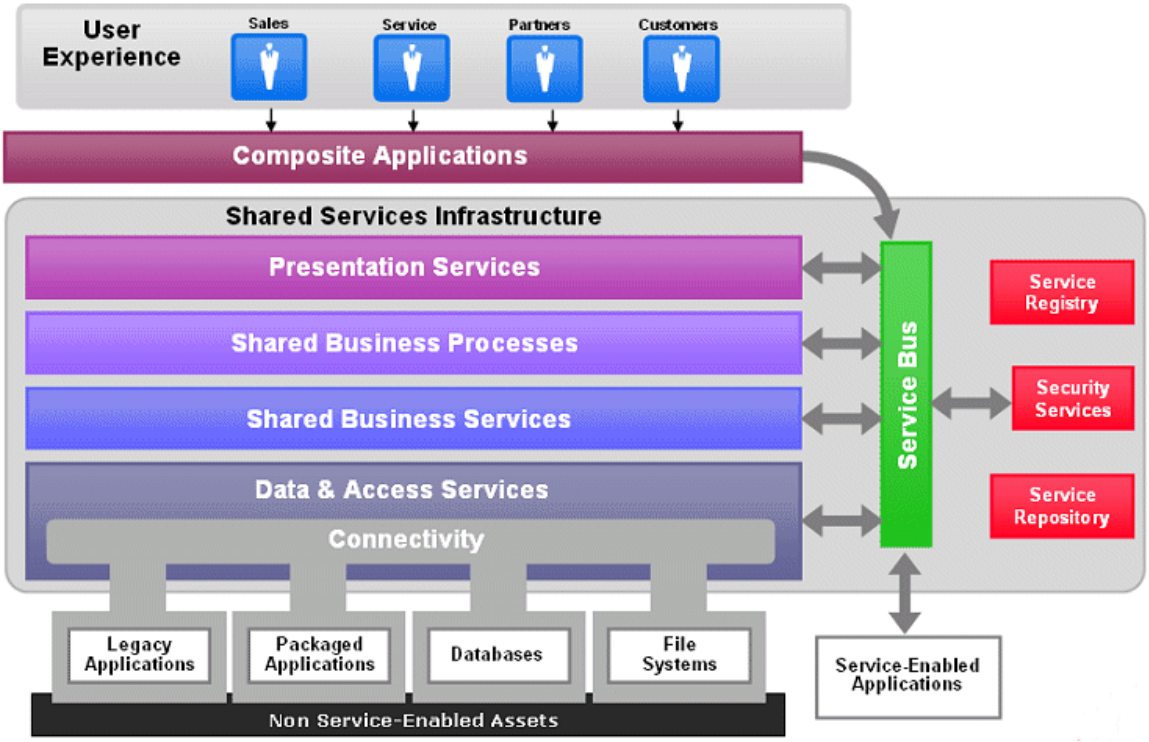

Figura 7: Arquitectura de un SOA.[7]

[7] Fuente: http://docs.oracle.com/

3.2 Principios de la arquitectura

La arquitectura SOA se asienta sobre los siguientes principios [11]:

1. **Contratos de servicio estandarizados**: Las interfaces de entrada y salida tienen que estar declaradas explícitamente, utilizando por ejemplo WSDL (Web Services Descripition Language).
2. **Servicios con bajo acoplamiento**: Necesario para una independencia mayor en el diseño del servicio y sus posteriores evoluciones.
3. **Abstracción**: Se ocultan los detalles internos del servicio, el cual debe ser una caja negra a la vista del cliente, que conoce solamente los detalles de la interfaz.
4. **Reusabilidad**: Reaprovechamiento de los servicios encapsulados para ser utilizados por otros servicios.
5. **Autonomía**: Control sobre la lógica de negocio encapsulada y el entorno de ejecución.
6. **Sin estado**: Todos los datos que necesita el servicio para funcionar tienen que provenir de parámetros de entrada.
7. **Capacidad de descubrimiento**: Gracias a los metadatos, el servicio puede ser descubierto de forma eficaz, utilizando para ello registros de servicios como UDDI.
8. **Composición**: Los servicios de más bajo nivel componen servicios complejos de más alto nivel.
9. **Interoperabilidad**: Entre servicios que se ejecutados en diferentes entornos y sistemas operativos.

3.3 Implementaciones basadas en SOA

3.3.1 Open Service Gateway Initiative (OSGi)

OSGi (*Open Service Gateway Initiative*) es la implementación utilizada en e-Gotham. OSGi son un conjunto de especificaciones abiertas que definen un sistema de componentes dinámicos para Java [12]. Estas especificaciones permiten un modelo de desarrollo donde las aplicaciones son creadas dinámicamente a partir de diferentes módulos, los cuales son reutilizables.

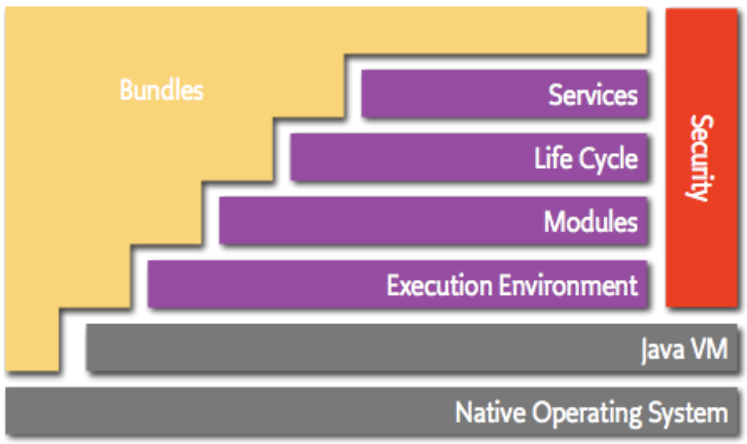

Figura 8: Arquitectura OSGi.[8]

[8] Fuente: http://www.osgi.org/

La funcionalidad de las capas de la arquitectura OSGi es la siguiente:

- **Bundles**: Son los componentes OSGi programados por los desarrolladores (módulos). Cada módulo va empaquetado en un fichero .jar.
- **Services**: Capa de servicios que conecta los módulos de manera dinámica.
- **Life Cycle**: API para instalar, iniciar, detener, actualizar y desinstalar módulos sin tener que detener la Máquina Virtual de Java (JVM).
- **Modules**: Capa que define como un módulo puede importar y exportar código. Define las reglas de interoperabilidad entre los mismos y la arquitectura de carga de clases.
- **Security**: Capa encargada con los aspectos de seguridad. Provee una infraestructura para desplegar aplicaciones que deben ejecutarse en un entorno controlado. Tiene dos objetivos básicos: integridad y autenticación.
- **Execution Enviroment**: Define que métodos y clases están disponibles en una plataforma específica.

3.3.2 Jini

Otra conocida implementación basada en SOA es Jini [13], una API para Java desarrollada inicialmente por Sun Microsystems, aunque el proyecto ha sido transferido a la fundación Apache bajo el nombre de River. Se utiliza para la creación de un sistema distribuido de grupos federativos de usuarios y recursos, los cuales pueden ser *hardware* o *software*.

La arquitectura Jini está basada en servicios, que pueden ser utilizados por una persona, programa o dispositivo. Los servicios se comunican entre sí mediante RMI y pueden ser añadidos o eliminados dinámicamente. Para conocer los servicios disponibles se consulta el servicio de búsqueda – *Look Up Service* (LUS) – el cual mapea las interfaces que indican la funcionalidad de un servicio con el conjunto de objetos que implementan dicho servicio. Un sistema de concesiones negocia entre el cliente y el proveedor el acceso a un servicio durante un determinado tiempo.

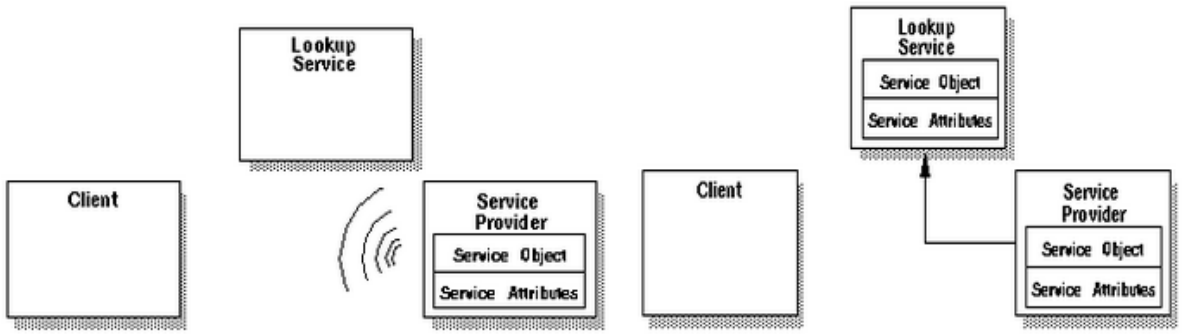

Figura 9: Proceso de búsqueda de un servicio en Jini.[9]

[9] Fuente: https://river.apache.org/

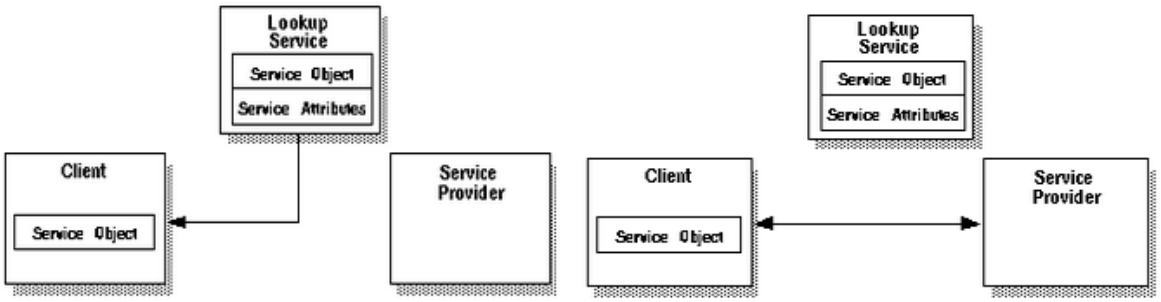

Figura 10: Proceso de registro de un servicio en Jini.[10]

3.4 Protocolos de descubrimiento y registro de servicios

3.4.1 Service Location Protocol (SLP)

SLP (Service Location Protocol) es un protocolo ideado por el *Service Location Protocol Working Group of Internet Engineering Task Force* (IETF) [14]. Es simple, extensible, escalable y descentralizado. Además, es independiente de la plataforma subyacente (la única restricción es que ha sido diseñado para redes IP, pero esto no es un problema debido al gran despliegue de estas redes). Los servicios están caracterizados por una lista de atributos (cada uno con su respectivo nombre y lista de posibles valores homogéneos). Las consultas de servicio están constituidas por una dupla atributo-valor. Pueden ser enviadas en *multicast* en una red local o en modo *unicast* a un servidor de directorio remoto en Internet, donde todos los servicios disponibles han sido previamente registrados.

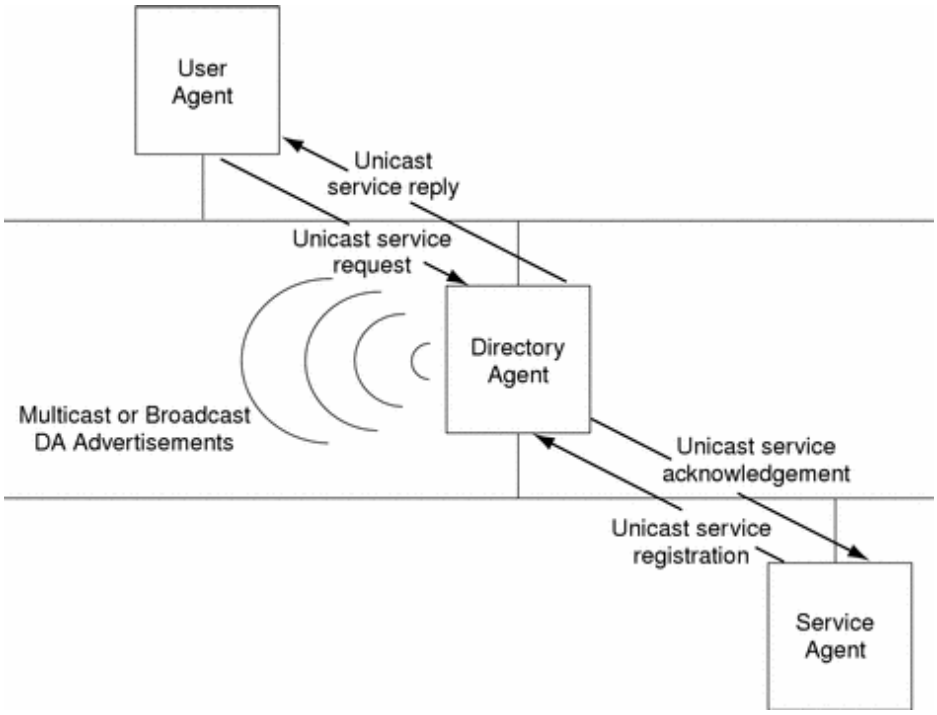

Figura 11: Implementación de la arquitectura SLP con un Agente de Directorio.[11]

[10] Fuente: https://river.apache.org/
[11] Fuente: http://docs.oracle.com/

3.4.2 Universal Plug and Play (UPnP)

Las tecnologías UPnP son un conjunto de estándares abiertos aceptados internacionalmente por la industria, que han simplificado la conectividad y usabilidad de dispositivos y servicios en el hogar. Estos estándares han estado operativos más de una docena de año, lo que sea traducido en la creación de numerosos dispositivos interoperables en el hogar, con millones de nuevos productos con tecnología UPnP distribuidos cada año [15].

La arquitectura de dispositivo UPnP es una plataforma de red abierta que aprovecha los protocolos existentes de Internet como IP, TCP, UDP y HTTP. Además, la arquitectura UPnP soporta transferencias de datos entre todos los dispositivos conectados a la red, mediante la definición de un mecanismo estandarizado para el descubrimiento y control de dispositivos utilizando XML (*eXtensive Markup Language*) y SOAP (*Simple Object Access Protocol*).

La arquitectura soporta el descubrimiento automático, por lo que un dispositivo puede unirse dinámicamente a una red, obtener una dirección IP, anunciar su nombre, transmitir sus capacidades bajo demanda, y aprender acerca de la presencia y características de otros dispositivos. Además, un dispositivo puede dejar una red sin dejar información de estado.

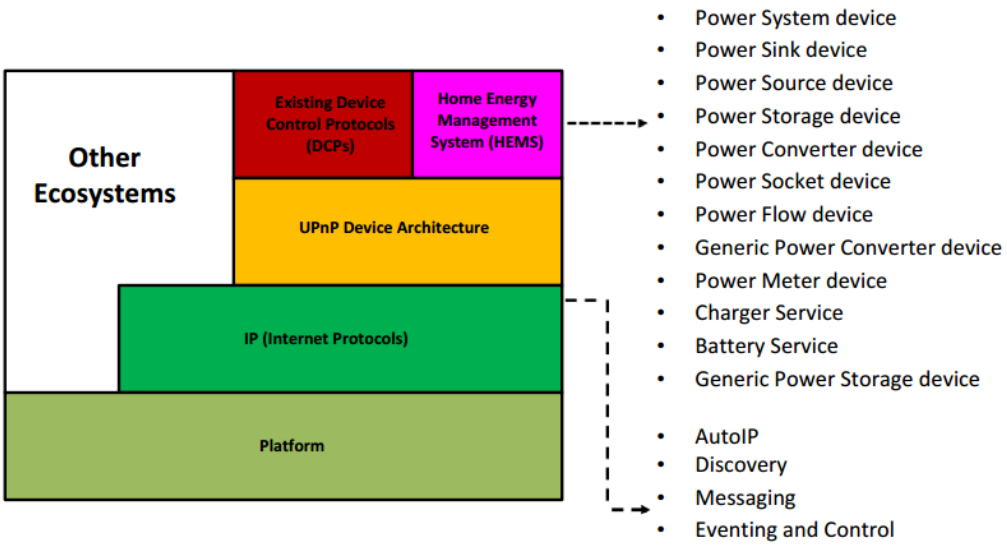

Figura 12: Ecosistema UPnP para soluciones de administración de energía.[12]

Existen muchas iniciativas en todo el mundo para desarrollar una red eléctrica inteligente para mejorar la eficiencia, reducir costes y proporcionar funciones de administración de energía. UPnP puede desempeñar un papel en la gestión de energía proporcionando una arquitectura de comunicaciones común dentro de la HAN (*Home Area Network*).

El actual despliegue de productos basados en UPnP supone un ecosistema perfecto para el desarrollo y soporte de soluciones de administración de energía, ya que incorpora mecanismos de seguridad y descubrimiento de dispositivos. Además, soporta la integración con múltiples protocolos de Smart Grids, como SEP 2.0, NAESB's ESPI Initiative, OpenADR y OASIS, proveyendo una plataforma interoperable que permite la comunicación entre dispositivos y aplicaciones de gestión de energía, con mensajes de alertas o lecturas.

[12] Fuente: http://www.upnp.org/

3.4.3 Simple Service Discovery Protocol (SSDP)

SSDP (*Simple Service Discovery Protocol*) [16] es un protocolo de red basado en protocolos del Internet Protocol Suite (utiliza HTTP y UDP principalmente) para el anuncio y descubrimiento de servicios de red. Esto se logra sin la ayuda de mecanismos de configuración basados en servidor, como DHCP (*Dynamic Host Configuration Protocol*) o DNS (*Domain Name System*), y sin necesidad de una configuración estática. SSDP es la base del protocolo UPnP y su uso está pensado para entornos pequeños.

SSDP utiliza el método `NOTIFY HTTP` para anunciar el establecimiento o supresión de los servicios mediante *multicast*. Un cliente que desee descubrir los servicios disponibles en una red, utilizará el método `M-SEARCH`.

Las respuestas a las dichas solicitudes se envían a través de *unicast* a la dirección origen y número de puerto de la solicitud de *multicast*. SSDP puede ser utilizado sobre IPv4 o IPv6.

3.4.4 Universal Description, Discovery, and Integration (UDDI)

UDDI (*Universal Description, Discovery, and Integration*) es un registro basado en XML donde las empresas de todo el mundo pueden publicarse en Internet, y un mecanismo para registrar y localizar aplicaciones de servicios web [17]. UDDI es una iniciativa industrial abierta, impulsada por la *Organization for the Advancement of Structured Information Standards* (OASIS) y fue propuesto como un estándar del núcleo de los *Web Service*. Está diseñado para ser interrogado mediante mensajes SOAP y para facilitar el acceso a documentos WDSL, que describen los formatos de mensajes necesarios para interactuar con los servicios web que figuran en su directorio.

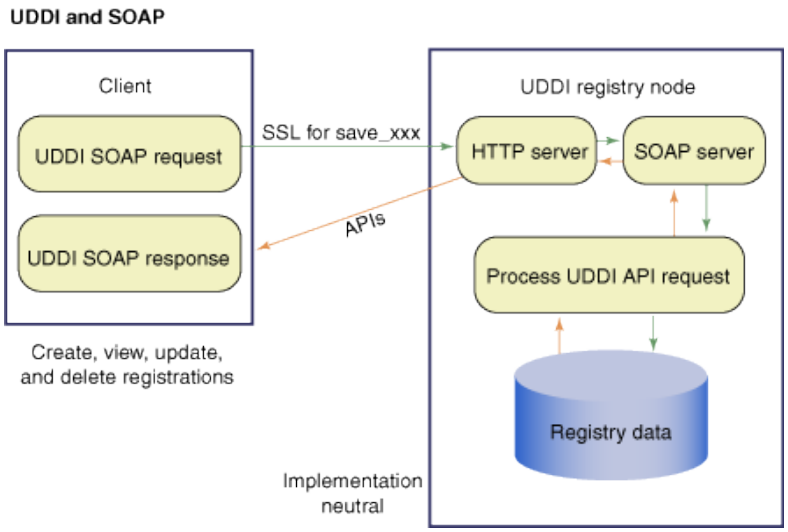

Figura 13: Proceso de registro de un servicio web en UDDI.[13]

[13] Fuente: http://uddi.xml.org/

4 Ontologías de servicios

4.1 ¿Qué es una ontología?

Una ontología define los términos usados para describir y representar un área de conocimiento. Las ontologías son usadas por personas, bases de datos y aplicaciones que necesitan compartir información de dominio (tema específico). Las ontologías incluyen definiciones de conceptos básicos del dominio y las relaciones entre ellas. Básicamente, codifican conocimiento y hacen que este sea reutilizable [18].

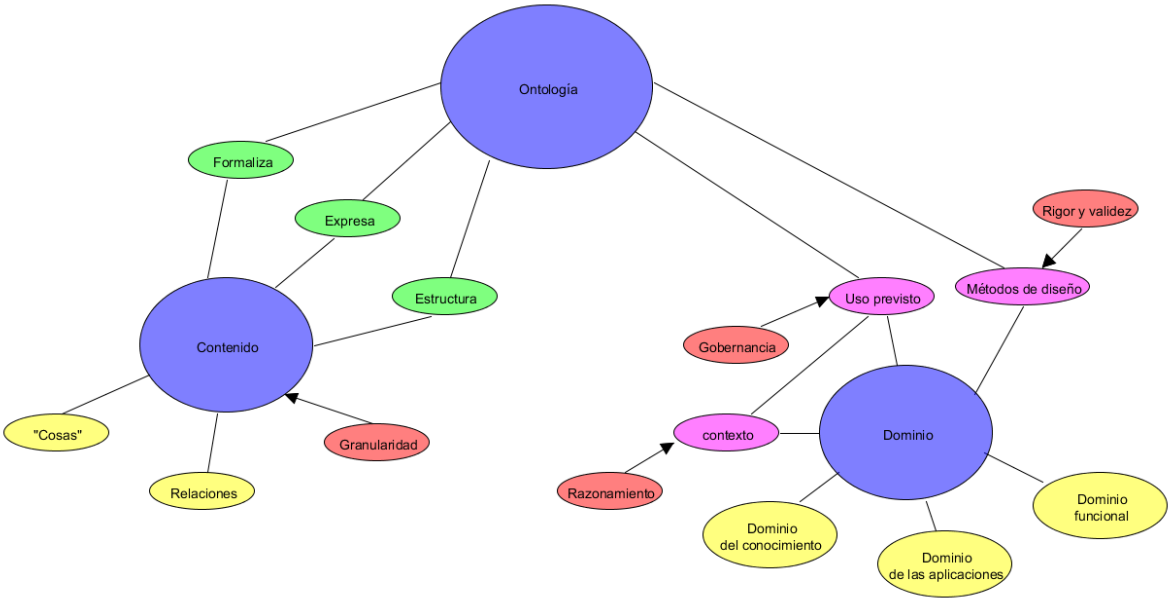

Figura 14: Visión general de una ontología.

La palabra ontología ha sido utilizada para describir objetos con diferentes grados de estructura. Estos van desde simples taxonomías hasta esquemas de metadatos o teorías lógicas. Dentro de estos grados de estructura, encontramos los siguientes conceptos:

- Clases (cosas generales) de dominios de interés variados.
- Relaciones entre las cosas.
- Propiedades o atributos de las cosas.

Las ontologías se expresan normalmente en un lenguaje basado en la lógica, para que puedan hacerse distinciones detalladas, precisas, coherentes y significativas entre las clases, propiedades y relaciones. Algunas herramientas de ontologías pueden realizar un razonamiento automatizado, ofreciendo servicios avanzados para aplicaciones tales como bases de datos inteligentes, consultas semánticas, soporte de decisiones, comprensión del lenguaje natural y gestión del conocimiento.

Las ontologías pueden ser muy útiles para estructurar y definir el significado de metadatos que actualmente están siendo recolectados y estandarizados. Las aplicaciones del futuro podrán ser "inteligentes", en el sentido de que podrán trabajar con mayor precisión en el nivel conceptual humano.

El uso de ontologías es esencial en aplicaciones que requieren buscar o combinar información de diversos dominios. Aunque el uso de XML DTD y XML *Schemas* son suficientes para el intercambio de datos entre dos entidades que han acordado las definiciones de antemano, su falta de semántica impide a las computadoras realizar esta tardea de forma fiable debido a nuevos vocabularios XML. El mismo término puede ser usado con (a veces sutil) diferente significado en diferentes contextos, y diferentes términos pueden referirse a objetos que tienen el mismo significado. RDF y RDF *Schema* comienzan a abordar este problema permitiendo asociar una semántica simple a los identificadores. Con RDF *Schema* se pueden definir clases que pueden tener múltiples subclases y superclases, las cuales pueden tener atributos, dominios y rangos. En este sentido, RDF *Schema* es un lenguaje simple de ontología. Sin embargo, con el fin de lograr la interoperabilidad entre diferentes esquemas desarrollados y gestionados de manera autónoma, es necesaria una semántica más amplia. Por ejemplo, RDF *Schema* no puede especificar que las clases Persona y Coche son disjuntas, o que un cuarteto de cuerda está integrado por cuatro músicos.

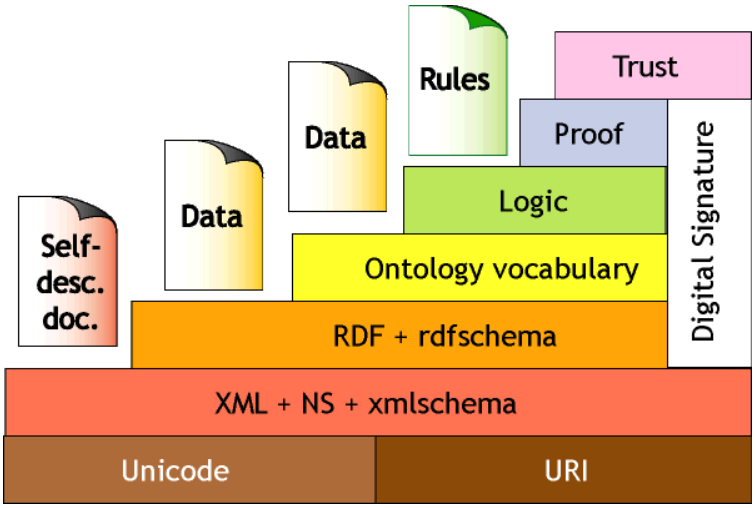

Figura 15: Arquitectura de una ontología de servicios.[14]

4.2 Lenguajes para la descripción de ontologías

4.2.1 Web Ontology Languague (OWL)

OWL (*Web Ontology Language*) está diseñado para ser usado en aplicaciones que necesitan procesar el contenido de la información en lugar de únicamente representar información para los humanos [19]. OWL facilita un mejor mecanismo de interpretabilidad de contenido Web que los mecanismos admitidos por XML, RDF, y RDF *Schema* (RDF-S) proporcionando vocabulario adicional junto con una semántica formal.

[14] Fuente: http://www.hipertexto.info/

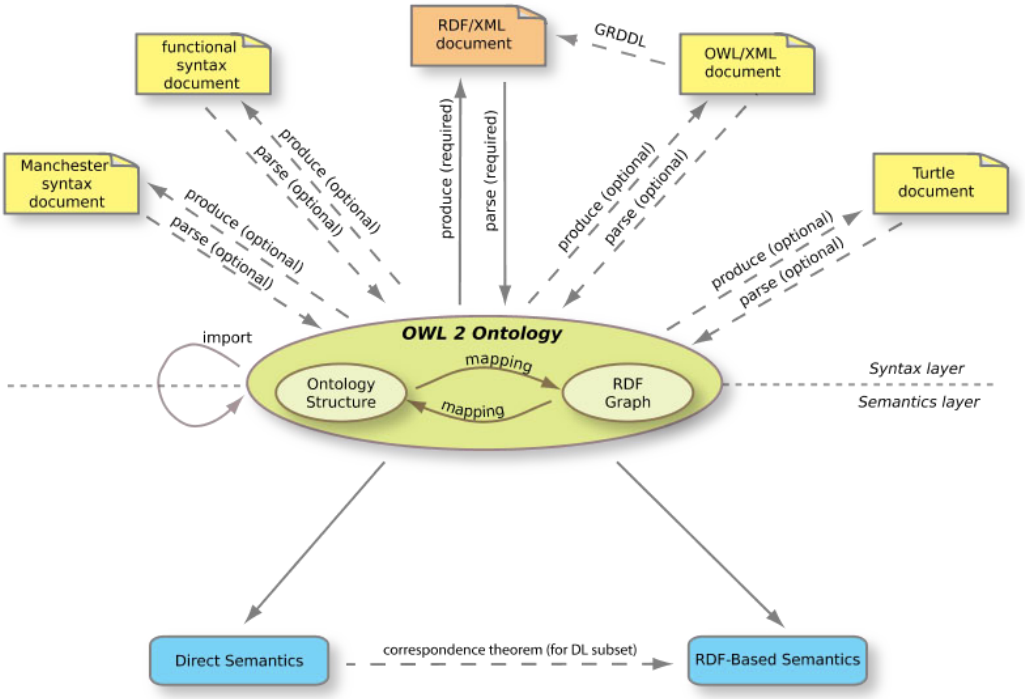

Figura 16: Estructura de OWL.[15]

4.2.2 Web Service Modeling Language (WSML)

WSML (*Web Service Modeling Language*) es un lenguaje de modelado de servicios web que provee una sintaxis formal y una semántica para WSMO (*Web Service Modeling Ontology*) [20]. Está basado en diversas lógicas formales:

- Lógicas descriptivas (*Description Logics*): familia de lenguajes de representación del conocimiento utilizados para representar conocimiento terminológico de un dominio de aplicación.
- Lógica de primer orden *(First-Order Logic*): sistema formal que estudia la inferencia de los lenguajes de primer orden.
- Programación lógica (*Logic Programming*): es un tipo de paradigma de programación dentro de la programación declarativa, basada en el concepto de predicado o relación entre elementos. Se utiliza para el modelado de *Web Services* semáticos.

También es posible el mapeo entre ontologías WSML y ontologías OWL para interoperar con ellas mediante un subconjunto semántico común.

4.2.3 Web Service Modeling Ontology (WSMO)

WSMO (*Web Service Modeling Ontology*) es un modelo conceptual basado en ontologías para la implementación e interoperabilidad de servicios web semánticos [21].

Se basa en WSMF (*Web Service Modeling Framework*) que separa los elementos que se necesitan para describir servicios en ontologías, objetivos y mediadores. WSML es el lenguaje usado para describir todos estos elementos. El modelo de servicios describe el

[15] Fuente: http://www.w3.org/

comportamiento del servicio en términos de procesos y de constructos de control. La base conecta procesos, entradas y salidas en el modelo de proceso a un determinado protocolo de transporte descrito en un documento de WSDL.

WSMO supone uno de los mayores esfuerzos para el propósito de especificar la información semántica de los *Web Services* para hacer posible la automatización de servicios, su composición y ejecución. WSMO recomienda el uso de vocabularios ampliamente aceptados tales como metadatos Dublin Core y FOAF (*Friend of a Friend*).

En WSMO, un objetivo especifica qué quiere el usuario y las descripciones del *Web Service* definen la capacidad que proporciona el servicio. Con WSMO también pueden expresarse las propiedades no funcionales del servicio.

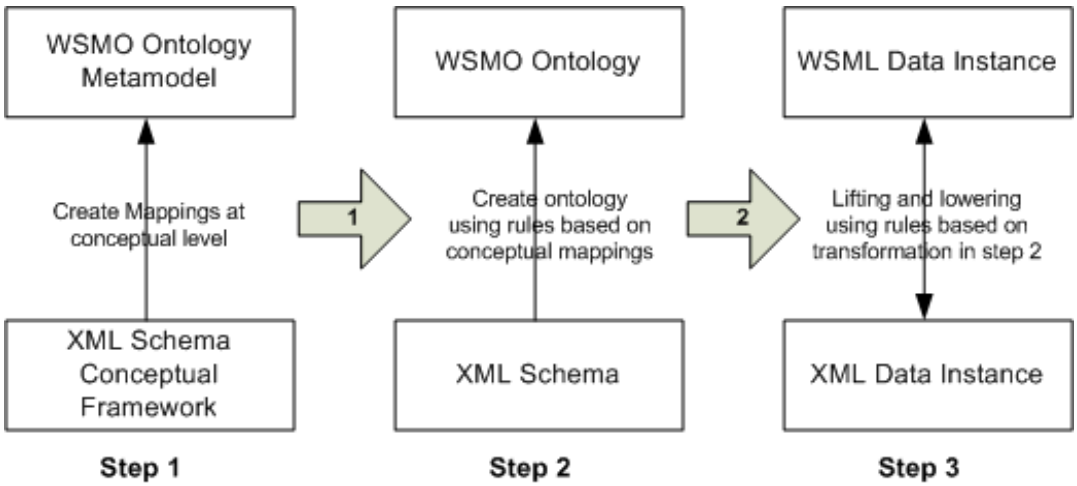

Figura 17: Capas del modelo de datos para WSMO y XML.[16]

4.3 Modelos de datos

4.3.1 Estándares IEC 61970 y 61968

El *National Institute of Standars and Technology* (NIST) define la transformación de las redes de energía actuales en las redes de energía del futuro en forma de un proceso llamado *Smart Grids*, que estipula cambios en la arquitectura de empresa (EA) que afectan a todos los estamentos: desde la capa de negocios hasta la de infraestructuras IT. La monitorización, protección y optimización automática de las operaciones de los elementos interconectados son cuestiones claves en las redes del futuro. Estos elementos incluyen desde la central generadora, redes de alto voltaje, instalaciones de almacenamiento de energía, vehículos eléctricos y electrodomésticos [22].

Los diferentes dominios (clientes, mercados, proveedores de servicios, generación, transmisión y distribución) pueden esperar beneficios en los campos de las finanzas, seguridad, eficiencia energética y conservación del medio ambiente.

Como resultado de este proceso de transición van a surgir muchos desafíos. Uno de ellos relacionado con las comunicaciones y el intercambio de datos donde varios estudios e informes abogan por el uso de estándares IEC 61970 y 61968 de la *International*

[16] Fuente: http://www.w3.org/

Electrotechnical Commission (IEC) como base sintáctica y semántica para la creación de arquitecturas orientadas a servicios.

4.3.2 Common Information Model (CIM)

Los estándares mencionados anteriormente proponen un modelo de información común, el CIM (*Common Information Model*), cuyo objetivo es permitir a las aplicaciones software intercambiar información sobre la configuración y el estado de la red eléctrica. El CIM consiste en un modelo UML que define un vocabulario común y una ontología básica para los aspectos de la industria eléctrica. El núcleo del CIM es el *Wires Model*, que describe los componentes básicos utilizados en el transporte de electricidad.

El estándar que define los paquetes básicos del CIM es el IEC 61970-301, el cual se centra en las necesidades en la transmisión de electricidad, donde se incluyen aplicaciones relacionadas con los sistemas de gestión, planificación y optimización de energía (sistemas SCADA). Los estándares IEC 61970-501 y 61970-452 definen un formato XML para el intercambio de modelos de red utilizando RDF (Resource Description Framework).

Los estándares IEC 61968 amplían el CIM para satisfacer las necesidades de la distribución eléctrica, donde se incluyen aplicaciones relacionadas con:

- Sistema de gestión de la distribución.
- Sistema de gestión de incidencias.
- Planificación, medición y gestión del trabajo.
- Sistemas de información geográfica.
- Gestión de activos.
- Sistemas de información al cliente.
- Planificación de recursos empresariales.

4.3.3 Resource Description Framework (RDF)

RDF es una familia de especificaciones de la *World Wide Web Consortium* (W3C) originalmente diseñado como un modelo de datos para metadatos. Puede ser usado como un método general para la descripción conceptual o modelado de la información que se implementa en los *Web Services*, utilizando una variedad de notaciones de sintaxis y formatos de serialización de datos.

El modelo de datos RDF es similar a los enfoques de modelado conceptual clásicos como entidad-relación o diagramas de clases, ya que se basa en la idea de hacer declaraciones sobre los recursos en forma de expresiones sujeto-predicado-objeto – llamadas tripletas - donde el sujeto indica el recurso y el predicado denota aspectos del recurso y expresa una relación entre el sujeto y el objeto.

Este mecanismo para describir recursos es un importante componente de la actividad de la Web Semántica del W3C mediante el que el software automatizado puede almacenar, intercambiar y utilizar información legible por computadoras distribuidas a través de la Web, lo que a su vez, permite a los usuarios manejar la información con mayor eficiencia y seguridad.

El modelo de datos simple de RDF y la capacidad de modelar diferentes conceptos abstractos ha llevado a su uso en aplicaciones de gestión del conocimiento no relacionadas con la Web Semántica.

Una colección de sentencias RDF representa intrínsecamente un multigrafo dirigido y etiquetado. Como tal, un modelo de datos basado en RDF se adapta de forma más natural a cierto tipo de representación del conocimiento que el modelo relacional y otros modelos ontológicos.

4.3.4 Modelado de un circuito eléctrico con CIM

En la Figura 19 se muestra la jerarquía de clases para describir componentes CIM:

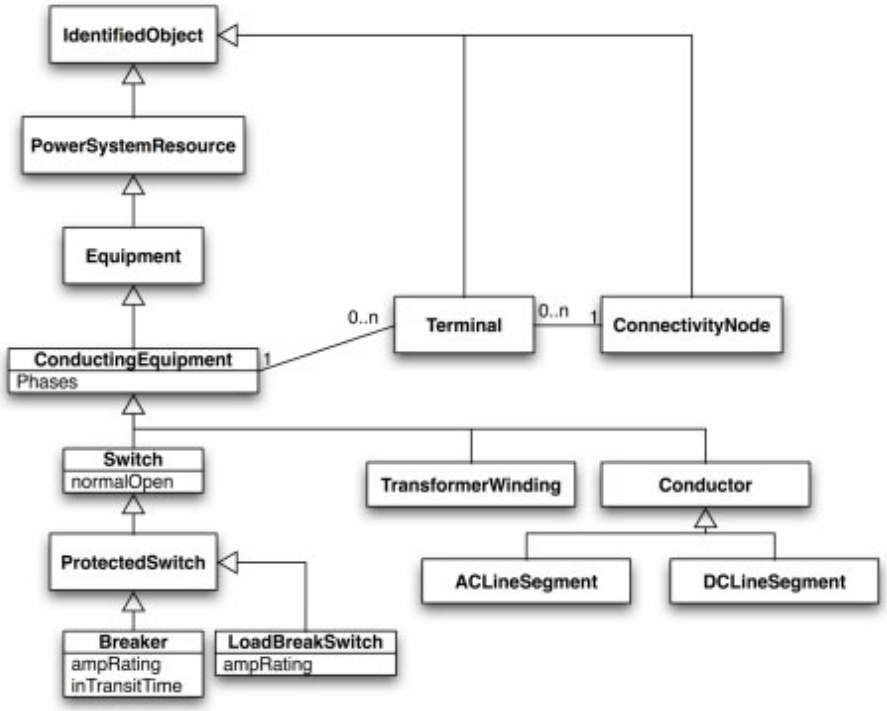

Figura 18: Diagrama de clases de equipamiento de suministro.[17]

En esta figura se muestra como son los *Connectivity Nodes* y *Terminals* que son usados para definir la interconexión de componentes dentro de la red:

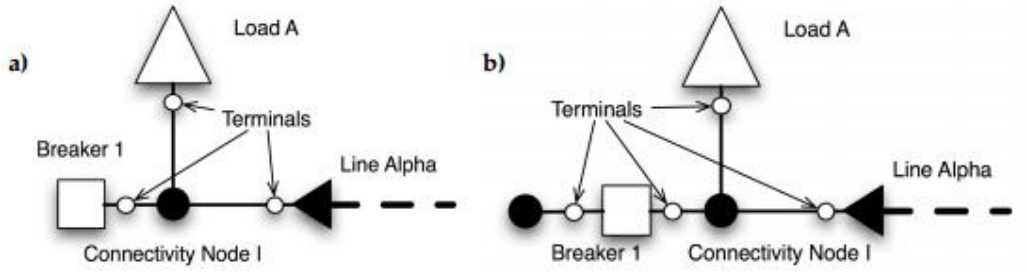

Figura 19: Ejemplo de conectividad entre un nodo y los terminales.[18]

[17] Fuente: A. W. Mcmorran. *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model.*
[18] Fuente: A. W. Mcmorran. *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model.*

A continuación, a modo de ejemplo, se muestra como modelar en objetos CIM los niveles de voltaje, transformadores y generadores de corriente a partir de un diagrama lineal:

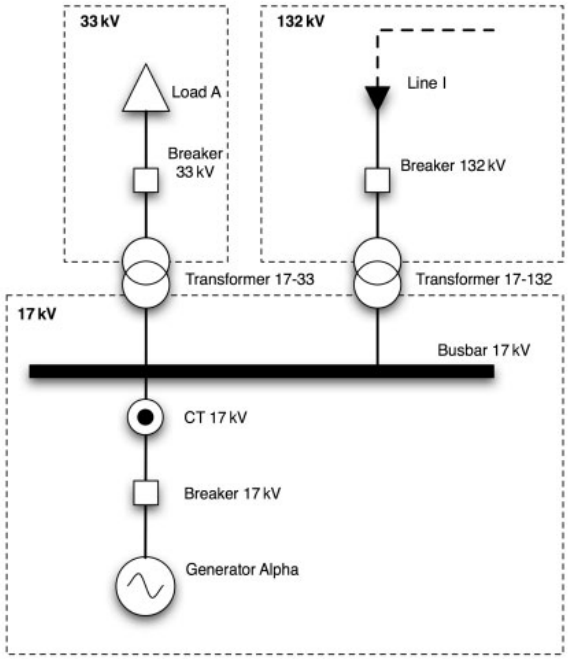

Figura 20: Diagrama lineal de un circuito eléctrico.[19]

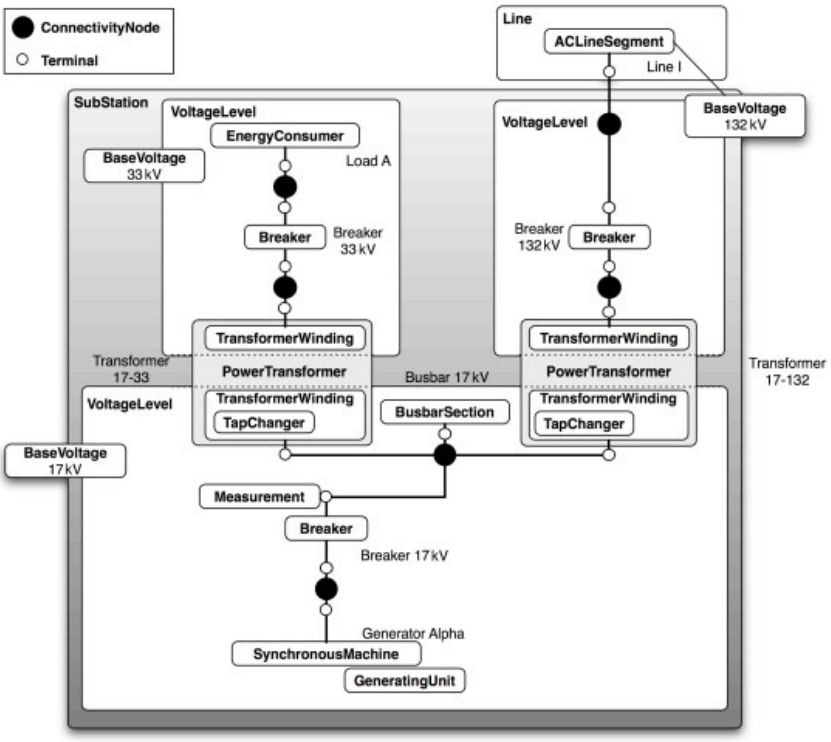

Figura 21: Diagrama de componentes de un circuito eléctrico.[20]

[19] Fuente: A. W. Mcmorran. *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model.*
[20] Fuente: A. W. Mcmorran. *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model.*

4.3.5 Ejemplo CIM RDF XML

A continuación se muestra el caso de un transformador y como sería su representación CIM RDF XML:

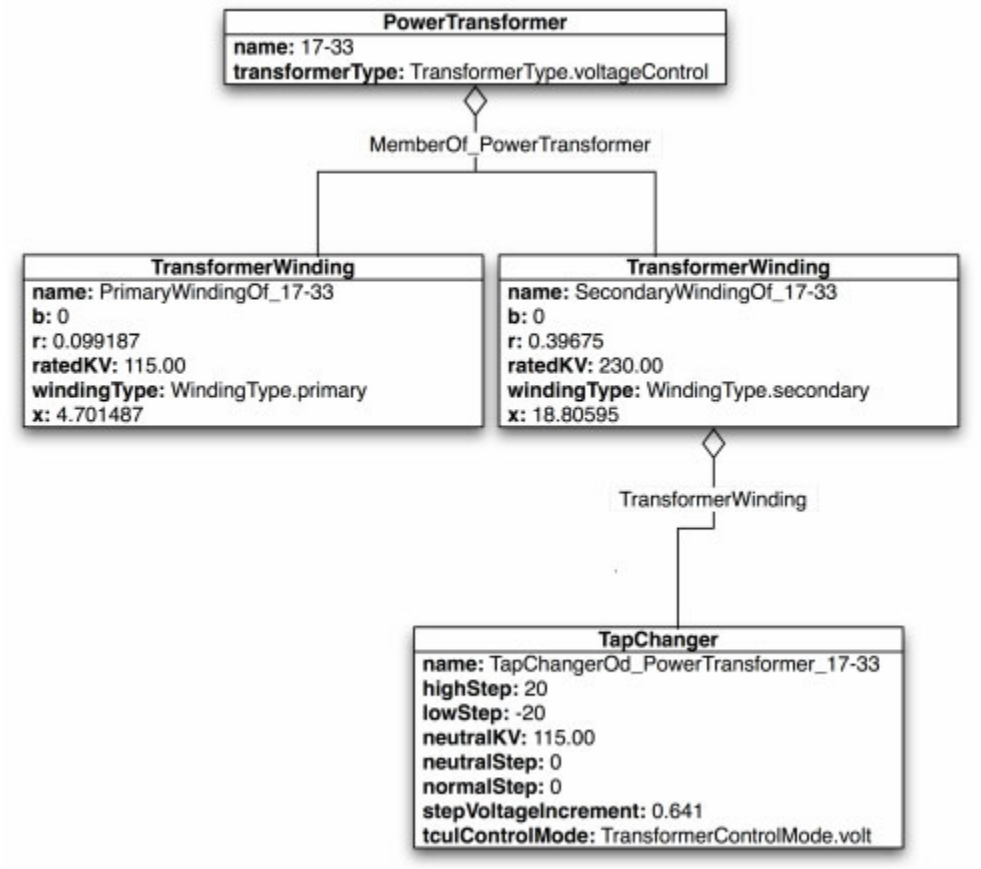

Figura 22: Transformador modelado en CIM.[21]

Se observa como de un objeto principal heredan otros que se especializan y amplían la funcionalidad, de un modo parecido como sucede en los lenguajes de programación orientados a objetos. En la figura superior tenemos un transformador de energía genérico del que heredan dos transformadores de bobina, uno de los cuales tiene un conmutador asociado.

[21] Fuente: A. W. Mcmorran. *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model.*

```
<cim:TransformerWinding rdf:ID="SecondaryWindingOf_PowerTransformer_1733">
   <cim:TransformerWinding.b>0</cim:TransformerWinding.b>
   <cim:TransformerWinding.r>0.39675</cim:TransformerWinding.r>
   <cim:TransformerWinding.ratedKV>230.00</cim:TransformerWinding.ratedKV>
   <cim:TransformerWinding.windingType
rdf:resource="http://iec.ch/TC57/2003/CIM-schema-
cim10#WindingType.secondary"/>
   <cim:TransformerWinding.x>18.80595</cim:TransformerWinding.x>
   <cim:TransformerWinding.MemberOf_PowerTransformer
rdf:resource="#PowerTransformer_302"/>
   <cim:Naming.name>SecondaryWindingOf_17-33</cim:Naming.name>
</cim:TransformerWinding>

<cim:TapChanger rdf:ID="TapChangerOf_PowerTransformer_1733">
   <cim:TapChanger.highStep>20</cim:TapChanger.highStep>
   <cim:TapChanger.lowStep>-20</cim:TapChanger.lowStep>
   <cim:TapChanger.neutralKV>115.00</cim:TapChanger.neutralKV>
   <cim:TapChanger.neutralStep>0</cim:TapChanger.neutralStep>
   <cim:TapChanger.normalStep>0</cim:TapChanger.normalStep>
   <cim:TapChanger.stepVoltageIncrement>0.641</cim:TapChanger.stepVoltageIncre
ment>
   <cim:TapChanger.tculControlMode  rdf:resource="http://iec.ch/TC57/2003/CIM-
schema-cim10#TransformerControlMode.volt"/>
   <cim:TapChanger.TransformerWinding
rdf:resource="#PrimaryWindingOf_PowerTransformer_302"/>
   <cim:Naming.name>TapChangerOf_PowerTransformer_17-33</cim:Naming.name>
</cim:TapChanger>

</rdf:RDF>

<cim:TransformerWinding rdf:ID="SecondaryWindingOf_PowerTransformer_1733">
   <cim:TransformerWinding.b>0</cim:TransformerWinding.b>
   <cim:TransformerWinding.r>0.39675</cim:TransformerWinding.r>
   <cim:TransformerWinding.ratedKV>230.00</cim:TransformerWinding.ratedKV>
   <cim:TransformerWinding.windingType
rdf:resource="http://iec.ch/TC57/2003/CIM-schema-
cim10#WindingType.secondary"/>
   <cim:TransformerWinding.x>18.80595</cim:TransformerWinding.x>
   <cim:TransformerWinding.MemberOf_PowerTransformer
rdf:resource="#PowerTransformer_302"/>
   <cim:Naming.name>SecondaryWindingOf_17-33</cim:Naming.name>
</cim:TransformerWinding>

<cim:TapChanger rdf:ID="TapChangerOf_PowerTransformer_1733">
   <cim:TapChanger.highStep>20</cim:TapChanger.highStep>
   <cim:TapChanger.lowStep>-20</cim:TapChanger.lowStep>
   <cim:TapChanger.neutralKV>115.00</cim:TapChanger.neutralKV>
   <cim:TapChanger.neutralStep>0</cim:TapChanger.neutralStep>
   <cim:TapChanger.normalStep>0</cim:TapChanger.normalStep>
   <cim:TapChanger.stepVoltageIncrement>0.641</cim:TapChanger.stepVoltageIncre
ment>
   <cim:TapChanger.tculControlMode  rdf:resource="http://iec.ch/TC57/2003/CIM-
schema-cim10#TransformerControlMode.volt"/>
   <cim:TapChanger.TransformerWinding
rdf:resource="#PrimaryWindingOf_PowerTransformer_302"/>
   <cim:Naming.name>TapChangerOf_PowerTransformer_17-33</cim:Naming.name>
</cim:TapChanger>

</rdf:RDF>
```

Figura 23: Representación CIM XML RDF del transformador del ejemplo.[22]

[22] A. W. Mcmorran. *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model.*

4.4 *Frameworks* y lenguajes de consulta semánticos

4.4.1 Jena

Jena es un *framework* para Java de código abierto usado en la Web Semántica. Proporciona una API para extraer datos y escribirlos en grafos RDF [23]. Los grafos se representan como un modelo abstracto que puede ser originado con los datos de archivos, bases de datos, URLs o una combinación de estos. Jena proporciona soporte para OWL y tiene varios razonadores internos incluido el *Pellet reasoner* (Un razonador OWL-DL Java de código abierto).

El framework de Jena incluye:

- Una API para leer, procesar y escribir datos RDF en formato XML, N-tripletas y Turtle.
- Una API para el manejo de ontologías OWL y RDFS.
- Un motor de inferencia basado en reglas para razonar con fuentes de datos RDF y OWL.
- Almacenamiento eficiente en disco de un gran número de tripletas RDF.
- Un motor de consultas SPARQL.

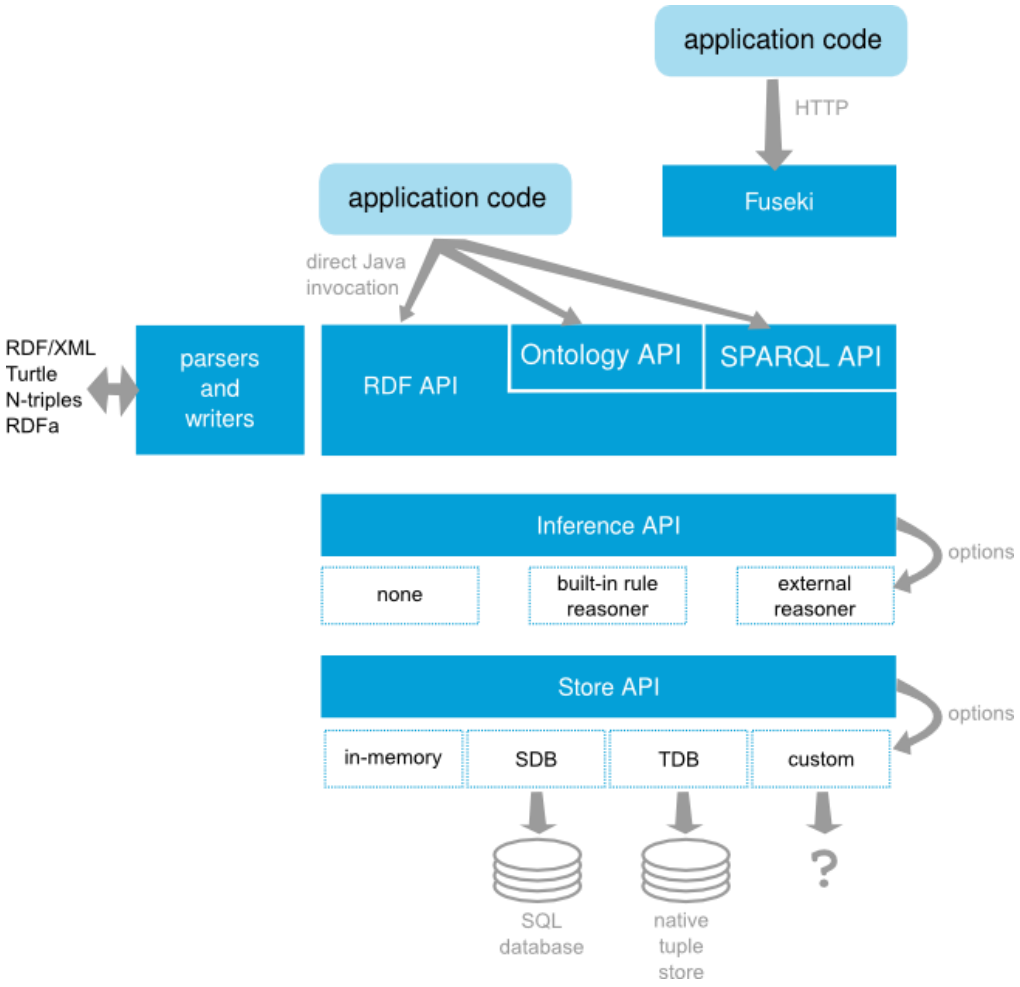

Figura 24: Arquitectura de Jena.[23]

[23] Fuente: https://jena.apache.org/

4.4.2 SPARQL

SPARQL (*SPARQL And RDF Query Language)* es un lenguaje de consulta de bases de datos, capaz de recuperar y manipular datos almacenados en formato RDF [24]. Fue hecho estándar por el *RDF Data Access Working Group* (DAWG) del W3C, y está reconocida como una tecnología clave de la web semántica.

SPARQL permite consultas basadas en tripletas, conjunciones, disyunciones y patrones opcionales. Existen implementaciones de SPARQL en múltiples lenguajes de programación: JAVA, Python, C, JavaScript, PHP, Perl y Ruby.

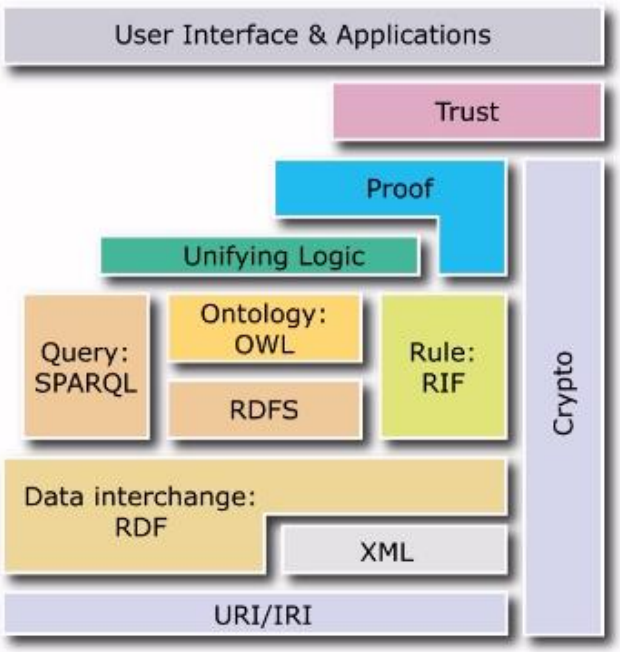

Figura 25: Arquitectura SPARQL/RDF.[24]

Las consultas pueden ser distribuidas a varios *SPARQL endpoints* – servicios que aceptan consultas SPARQL y devuelven resultados – un procedimiento conocido como *federated query*.

En caso de las consultas que leen datos de la base de datos, el lenguaje SPARQL especifica cuatro variaciones de consulta diferentes para diferentes propósitos:

- `SELECT`: Se utiliza para extraer valores brutos de un *SPARQL endpoint*. Los resultados se devuelven en formato de tabla.
- `CONSTRUCT`: Se utiliza para extraer la información desde el *SPARQL endpoint* y transformar los resultados en un RDF válido.
- `ASK`: Se utiliza para proporcionar una resultado Verdadero/Falso de una consulta a un *SPARQL endpoint*.
- `DESCRIBE`: se utiliza para extraer un grafo RDF desde el *SPARQL endpoint*, cuyo contenido se deja a criterio del administrador de la base de datos

[24] Fuente: http://www.w3.org/

5 *Aplicación Android*

5.1 Introducción

La aplicación desarrollada es una herramienta abierta para la creación de interfaces de gestión de información en *Smart Grids* mediante un dispositivo inteligente con sistema operativo Android.

La aplicación permite:

- Representar información relativa a los servicios y dispositivos registrados en una *Smart Grid*.
- Guardar, cargar y compartir por correo electrónico ficheros HTML con la información anterior.
- Representar en un mapa la ubicación de los dispositivos.
- Representar medidas (voltaje, temperatura, etc.) en tiempo real.
- Aplicar filtros por identificador de dispositivo, modelo o fabricante.
- Realizar consultas SPARQL a bases de datos semánticas.
- Guardar y cagar consultas SPARQL en ficheros de texto almacenados en la tarjeta SD.

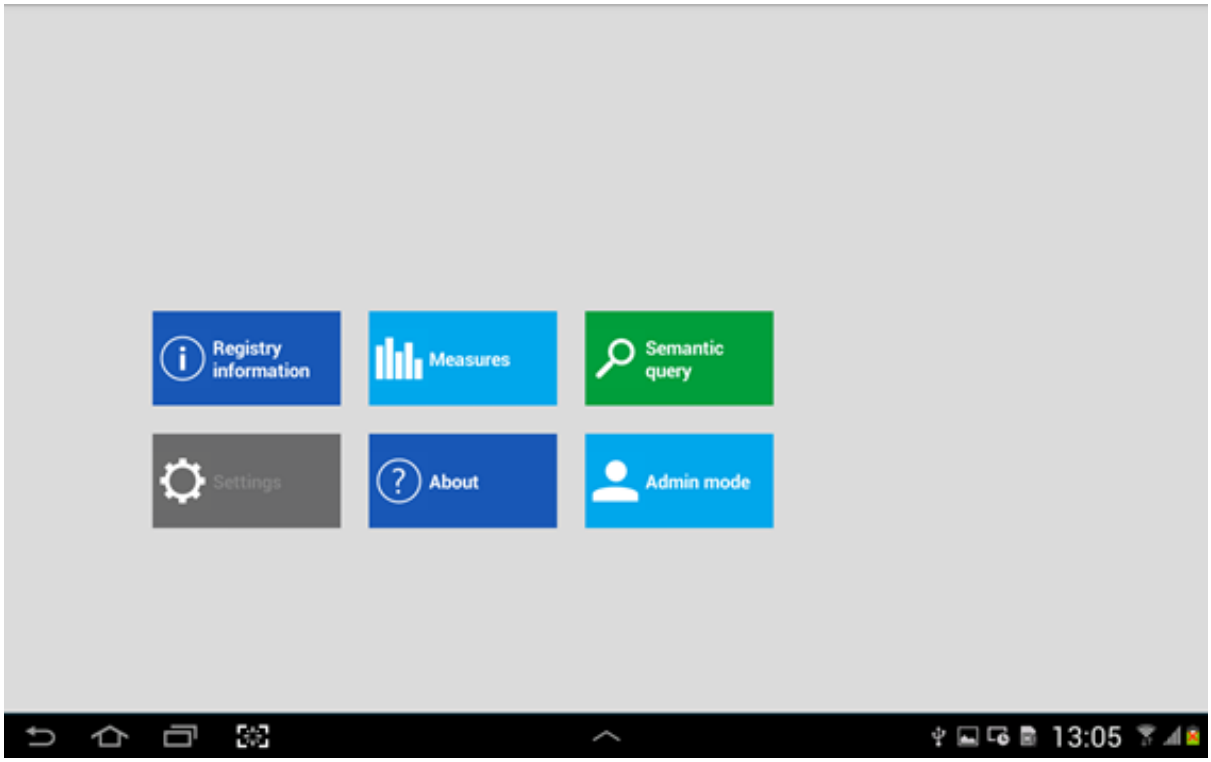

Figura 26: Pantalla principal de la aplicación.

5.2 Especificaciones

A continuación se detallan las especificaciones técnicas de la aplicación:

- Desarrollada para versión de Android 4.0 o superior en el dispositivo aunque es compatible con cualquier versión desde la 2.3.
- La interfaz gráfica ha de estar adaptada a todas las resoluciones de los diferentes dispositivos, y tendremos una vista distinta si utilizamos la aplicación desde un *smartphone* o una *tablet*, pero siempre con la misma funcionalidad en ambos.
- El código fuente de la aplicación estará escrito en JAVA y la interfaz gráfica en XML. Las imágenes utilizadas tienen licencia GLP y fueron obtenidas de un repositorio de imágenes *on-line*. El código va a ser liberado para que pueda ser modificado y adaptado a nuevas plataformas y nuevos servicios en el futuro.
- Es necesaria una conexión a Internet. En caso de conectarse mediante datos móviles, se recomienda una conexión 3G o superior.
- Se utilizará de una tarjeta de almacenamiento externa para guardar los ficheros HTML generados por la aplicación, los cuales pueden ser visualizados en cualquier dispositivo que soporte un navegador web.
- Se utilizará un servicio de mapas externo, *Google Maps*, para la representación de la ubicación de los dispositivos.
- La aplicación se conectará a diferentes interfaces RESTful para obtener la información de los dispositivos y servicios registrados en la *Smart Grid*.
- Soportará conexiones con cualquier *SPARQL endpoint* al que se tenga acceso para realizar cualquier consulta semántica.

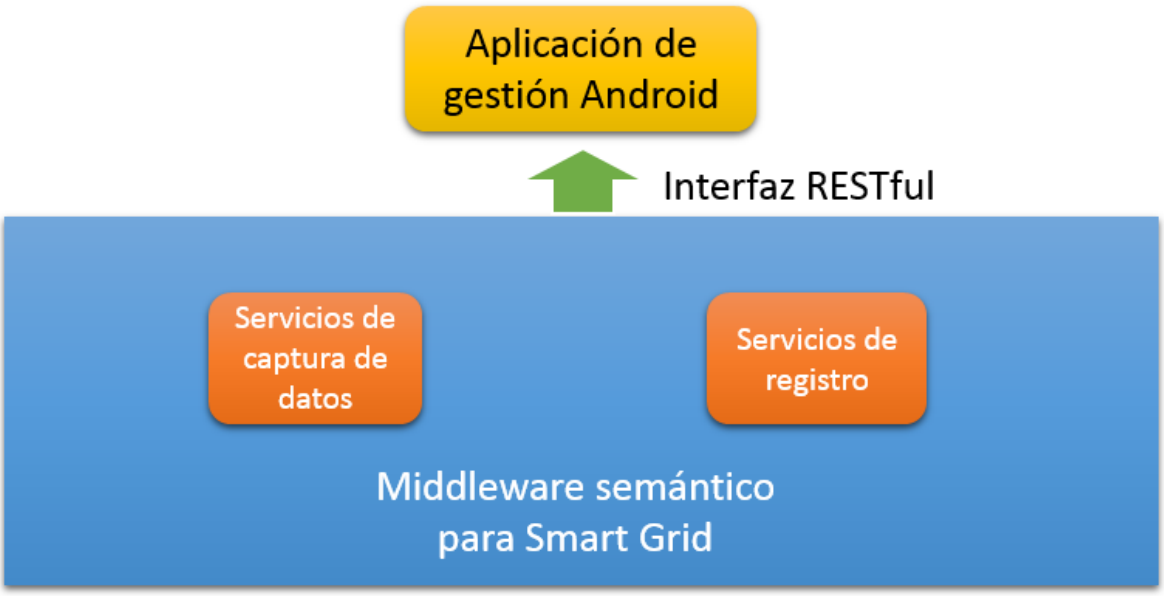

Figura 27: Integración de la aplicación con las interfaces RESTful.

5.3 Arquitectura de la aplicación

5.3.1 Diagrama de casos de uso

En la Figura 28 se muestra el diagrama de casos de uso de la aplicación modelado en UML. En él se muestran los 3 casos de uso principales:

- Representar información relativa a los servicios y dispositivos registrados.
- Representar medidas en tiempo real.
- Realizar consultas SPARQL a bases de datos semánticas.

En un siguiente nivel de profundidad se encuentran las diversas funcionalidades de la aplicación:

- Aplicar filtros por tipo de dispositivo, modelo o fabricante.
- Representar en un mapa la ubicación de los dispositivos.
- Cargar, guardar y compartir ficheros HTML con la información de registro.
- Cargar y guardar consultas SPARQL en ficheros de texto.

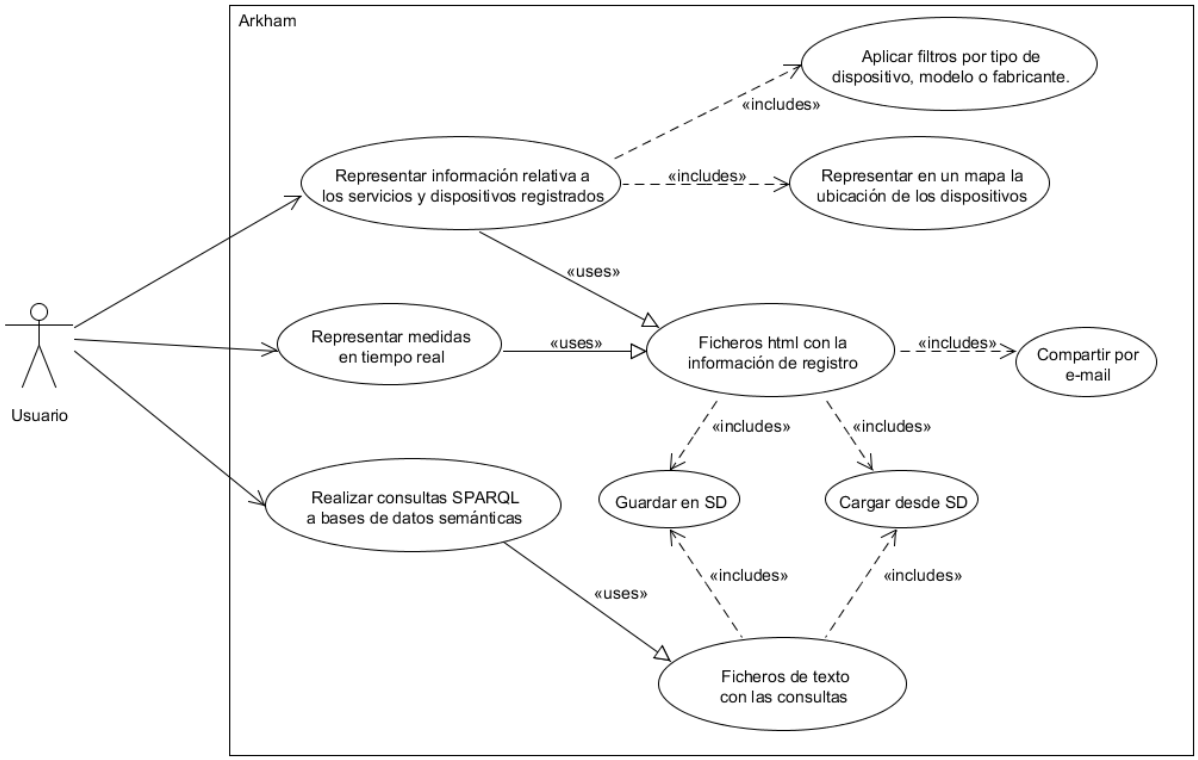

Figura 28: Diagrama de casos de uso.

5.3.2 Diagrama de clases

A continuación se muestra el diagrama de clases de la aplicación:

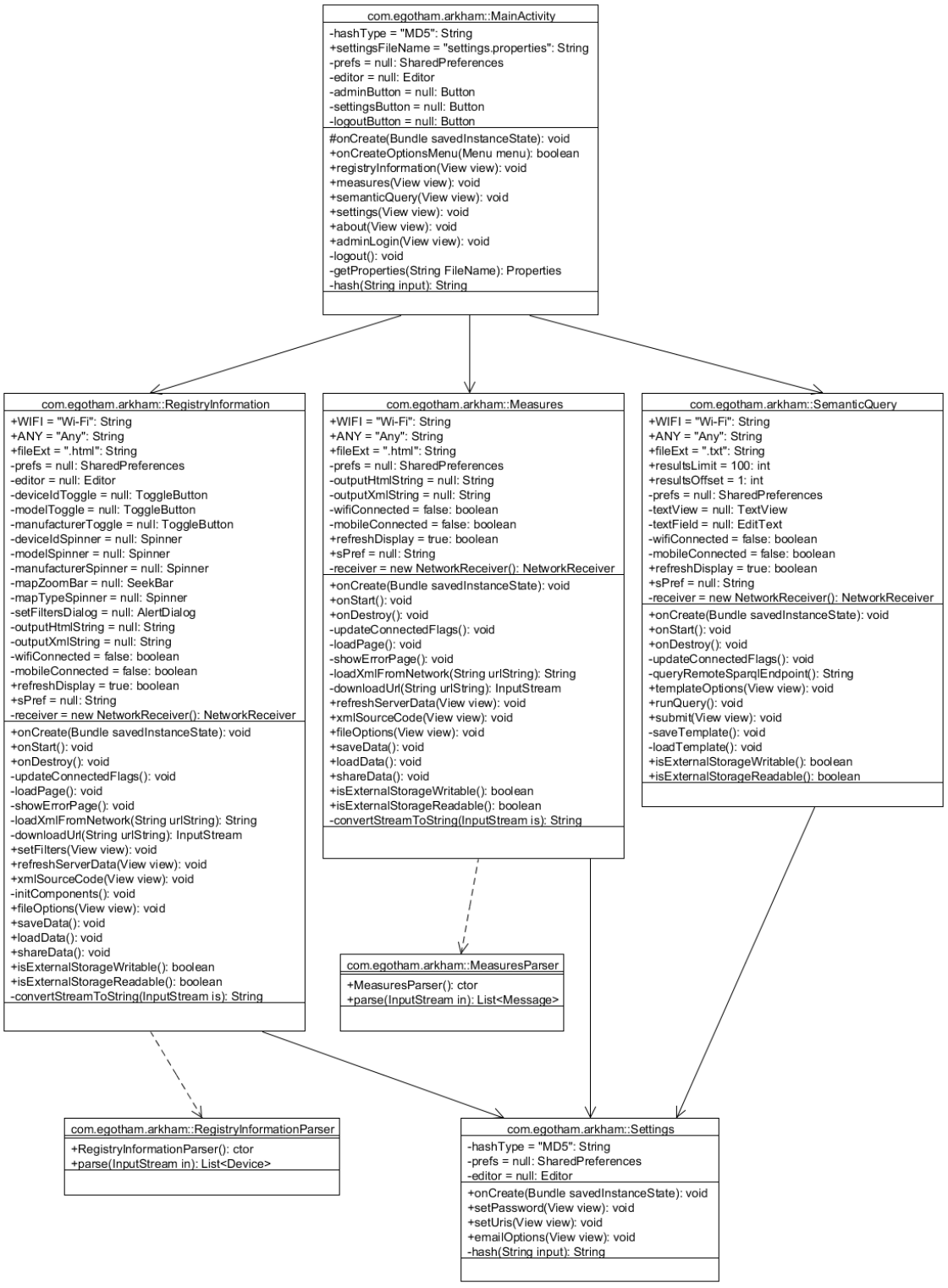

Figura 29: Diagrama de clases de la aplicación.

5.3.3 Diagrama de componentes

La Figura 30 detalla el diagrama de componentes de la aplicación:

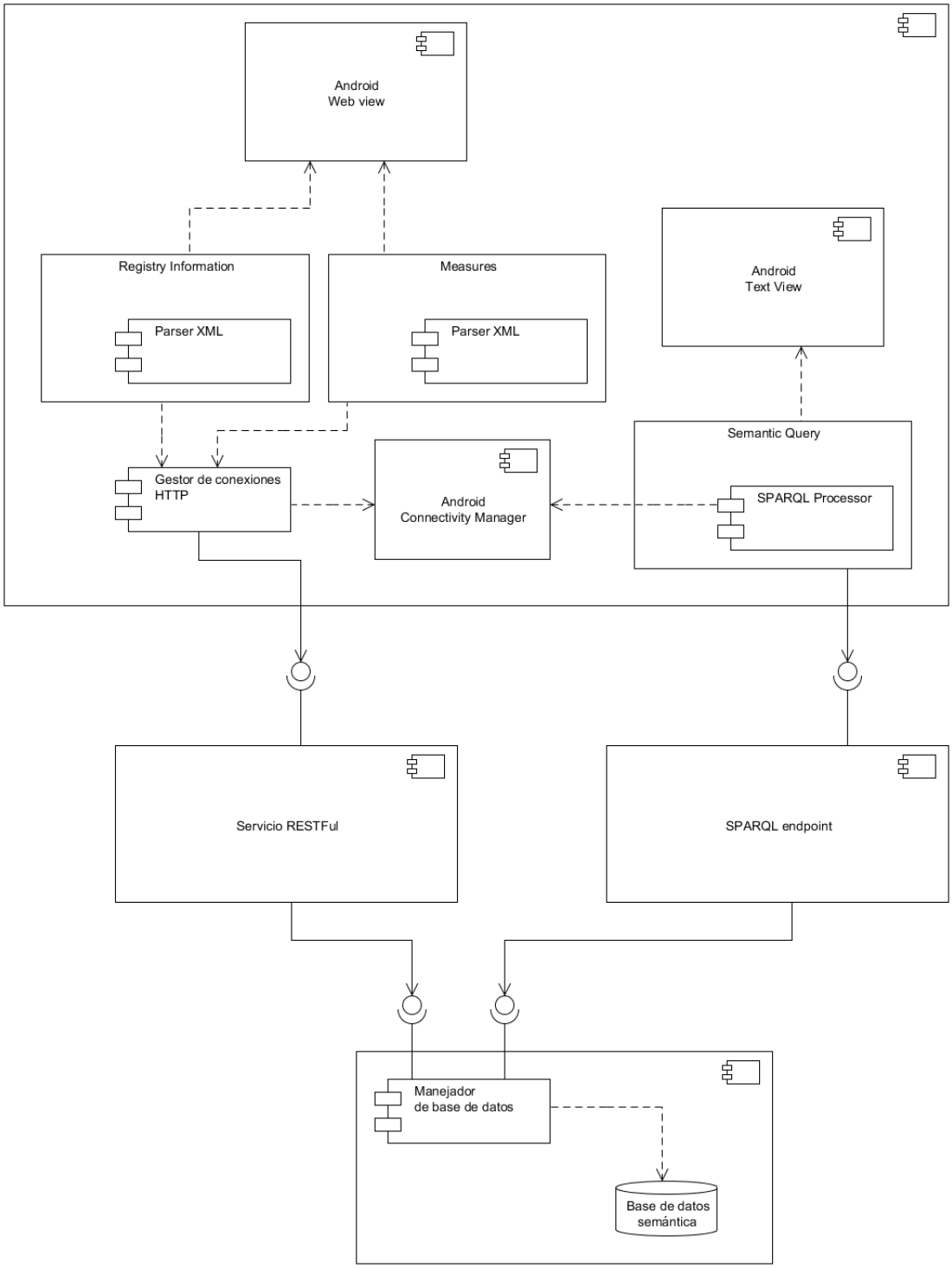

Figura 30: Diagrama de componentes de la aplicación.

5.3.4 Descripción de los componentes

Los componentes desarrollados, ilustrados en la Figura 30, tienen la siguiente funcionalidad:

- **Android Web View**: Componente Android que implementa un Navegador web embebido.
- **Android Text View**: Componente Android que implementa un visor de texto plano.
- **Android Connectivity Manager**: Componente Android que implementa un gestor de conectividad de red. Se encarga principalmente de comprobar el estado de la conexión (conectado/desconectado) y la interfaz de red utilizada (WiFi/datos móviles).
- **Parser XML**: Componente que implementa un parser XML adecuado al fichero que se espera recibir.
- **SPARQL Processor**: Componente que implementa un procesador de consultas SPARQL.
- **SPARQL endpoint**: Punto de acceso SPARQL al que es necesario conectarse para realizar las consultas semánticas.
- **Gestor de conexiones HTTP**: Componente que implementa el método "GET" de HTTP para descargar el fichero HTML del servicio RESTful.
- **Servicio RESTful:** Servicio que proporciona una interfaz a la que es posible conectarse mediante un gestor HTTP.
- **Manejador de base de datos**: Proceso encargado de servir de enlace entre las interfaces exteriores y el núcleo de la base de datos.

5.4 Detalles de implementación

5.4.1 Contraseña de administrador

Para el algoritmo de la contraseña del modo administrador se optó utilizar el mismo sistema empleado actualmente por un servidor web. Se guarda un resumen *hash* de la contraseña en el fichero "settings.properties", un fichero interno de la aplicación el cual no está accesible ni por el usuario, ni desde el exterior. De esta forma, se evita tener que guardar la contraseña en "claro". Si un atacante consiguiese obtener privilegios de *superuser* en nuestro dispositivo y llegase al fichero donde está almacenada la contraseña, lo único que encontraría sería el resumen *hash* de esta, lo que le sería inútil para poder acceder al modo administrador con esa información.

```
builder.setPositiveButton(R.string.login,
        new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                // Code for matching password
                String inputPassword = passwordField.getText()
                        .toString();
                String inputPasswordHash = hash(inputPassword);
                String passwordHash = prefs.getString(
                        Keys.passwordHashKey, Keys.blankValue);

                if (inputPasswordHash.equals(passwordHash)) {
                    Toast.makeText(getApplicationContext(),
                            R.string.admin, Toast.LENGTH_SHORT).show();
                    // Admin mode
                    logoutButton.setEnabled(true);
                    settingsButton.setEnabled(true);
                    settingsButton.setTextColor(Color.WHITE);
                    logoutButton.setVisibility(View.VISIBLE);
                    adminButton.setVisibility(View.GONE);
                } else {
                    Toast.makeText(getApplicationContext(),
                            R.string.wrong_password, Toast.LENGTH_SHORT)
                            .show();
                }
            }
        });
```

Figura 31: Implementación del diálogo del modo administrador.

En el código mostrado anteriormente, se observa cómo se genera el resumen *hash* de la contraseña introducida por el usuario y luego realiza una comparación con el *hash* que tiene guardado la aplicación. Si coinciden, se da acceso al modo administrador. En caso contrario, se muestra un mensaje de error al usuario.

```
/**
 * Returns the MD5 hash code of a string
 *
 * @param input
 *            - a string
 * @return the MD5 hash code
 */
private String hash(String input) {
    String hash = null;

    if (input != null) {
        try {
            MessageDigest digest = MessageDigest.getInstance(hashType);
            digest.update(input.getBytes(), 0, input.length());
            // Converts message digest value in base 16 (hex)
            hash = new BigInteger(1, digest.digest()).toString(16);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }

    return hash;
}
```

Figura 32: Implementación del algoritmo hash.

El código superior muestra el procedimiento utilizado para generar el resumen *hash*. La variable *hashType* contiene el nombre del algoritmo de resumen de utilizado. En este caso se ha utilizado MD5 por ser uno de los más habituales para esta casuística, aunque también podría usarse SHA o RIPEMD.

Para cambiar el algoritmo, simplemente habría que cambiar el código en la variable *hashType* que está definido como constante dentro de la clase "MainActivity":

```
private final String hashType = "MD5";
public final String settingsFileName = "settings.properties";
```

Figura 33: Parámetros configurables de la aplicación.

Algunos de los posibles códigos válidos:

- "MD2"
- "MD4"
- "SHA2"
- "SHA3"
- "RipeMD128"
- "RipeMD256"
- "RipeMD320"

5.4.2 Gestor de conexiones HTTP

Antes de poder acceder a Internet desde la aplicación, se necesitan declarar en el fichero "AndroidManifest.xml" los siguientes permisos:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Figura 34: Permisos de acceso a internet y al estado de la conexión de red.

El gestor de conexiones HTTP es el encargado de descargar el fichero XML del servicio RESTFul. Desde la versión 4.0 de Android, solo se permite realizar conexiones HTTP dentro de una clase que extienda de "AsyncTask" [25].

Esta clase dispone de 4 métodos que tienen que ser sobrescritos por el programador:

- **`onPreExecute()`**: Se invoca el en proceso de interfaz de usuario (*UI thread*) antes de que se ejecute la tarea. Este paso se utiliza normalmente para configurar la tarea, por ejemplo, mostrando una barra de progreso al usuario.
- **`doInBackGround()`**: Se invoca en el proceso de segundo plano (*Background* thread) inmediatamente después de que el método "onPreExecute" termine su ejecución. Este paso se usa para realizar una tarea que puede llevar mucho tiempo – como la descarga de un fichero desde un servidor – liberando así al proceso de interfaz de usuario (el cual se ejecuta en primer plano), lo que evita el bloqueo de la aplicación.
- **`onProgressUpdate()`**: La implementación de este método es opcional. Se invoca en el proceso de interfaz de usuario y sirve para actualizar el valor de la barra de progreso que se muestra al usuario.
- **`onPostExecute()`**: Se invoca el en proceso de interfaz de usuario después de que el método "doInBackGround" complete su tarea y cuyo resultado se pasa como parámetro en este paso.

```
/**
 * Implementation of AsyncTask used to download the XML file
 */
private class DownloadXmlTask extends AsyncTask<String, Void, String> {
    ProgressDialog progressDialog = new ProgressDialog(
            RegistryInformation.this, ProgressDialog.STYLE_SPINNER);

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog.setMessage(getResources()
                .getString(R.string.loading));
        progressDialog.show();
    }

    @Override
    protected String doInBackground(String... urls) {
        try {
            return loadXmlFromNetwork(urls[0]);
        } catch (IOException e) {
            return getResources().getString(R.string.connection_error);
        } catch (XmlPullParserException e) {
            return getResources().getString(R.string.xml_error);
        }
    }

    @Override
    protected void onPostExecute(String result) {
        setContentView(R.layout.activity_registry_information);
        // Displays the HTML string in the UI via a WebView
        WebView webView = (WebView) findViewById(R.id.activity_registry_information_web_view);
        webView.loadData(result, "text/html", null);
        progressDialog.dismiss();
    }
}
```

Figura 35: Implementación de una clase que extiende de la clase AsyncTask.

Como se puede observar en el código anterior, en el método “onPreExecute” se configura el diálogo de progreso que se muestra al usuario. A continuación, dentro del método “doInBackGround”, donde se encarga de descargar el fichero XML desde el servidor y generar el fichero HTML con el resultado, lo cual puede conllevar entre 1 y 3 segundos, dependiendo de la conexión que dispongamos en ese momento (WiFi o datos móviles). Utilizando WiFi, el resultado se muestra en aproximadamente un segundo.

En el siguiente código se muestra la utilización del gestor de conexiones HTTP utilizado, "HttpURLConnection", el cual utiliza el método **GET** de HTTP para obtener el fichero del servidor:

```
/**
 * Given a string representation of a URL, sets up a connection and gets an
 * input stream.
 *
 * @param urlString
 *            - URL of the XML.
 * @return an input stream
 * @throws IOException
 */
private InputStream downloadUrl(String urlString) throws IOException {
    URL url = new URL(urlString);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setReadTimeout(10000 /* milliseconds */);
    conn.setConnectTimeout(15000 /* milliseconds */);
    conn.setRequestMethod("GET");
    conn.setDoInput(true);
    // Starts the query
    conn.connect();
    InputStream stream = conn.getInputStream();
    return stream;
}
```

Figura 36: Implementación del método para descargar un fichero de un servidor web o servicio RESTful.

A continuación se muestra el extracto de código dentro del método "loadXmlFromNetwork" (que es invocado dentro de la clase "AsyncTask" como se ha visto anteriormente) donde se llama al método "downloadUrl" y cuyo resultado es parseado.

```
try {
    is = downloadUrl(urlString);
    devices = parser.parse(is);

    // Makes sure that the InputStream is closed after the app is
    // finished using it.
} finally {
    if (is != null) {
        is.close();
    }
}
```

Figura 37: Extracto de código donde se invoca el método downloadUrl

Una vez se haya realizado el parseo, es necesario cerrar la conexión con el servidor, ya que al tratarse de una conexión TCP, hay que liberar los recursos reservados durante la conexión.

5.4.3 Parser XML

Los dos *parsers* XML utilizados en la aplicación, uno para la clase "Registry Information" y otro para la clase "Measures", implementan la clase "XmlPullParser" de Android:

El funcionamiento es parecido a un *parser* SAX (Simple API for XML). Al igual que SAX, se trata de un *parser* orientado a eventos. A continuación se detalla como ejemplo el funcionamiento del *parser* utilizado para la clase "Measures".

Primero, se obtiene el tipo del primer evento que ocurre (un número entero). Éste se lanza cuando se lee la primera etiqueta del documento XML.

```
int eventType = parser.getEventType();
```

Figura 38: Implementación del parser XML (I).

A continuación, se repite un bucle hasta que ocurra el evento `END_DOCUMENT`. Cada vez que se lee una etiqueta, se lanza un evento `START_TAG`. Se guarda en una variable el nombre de la etiqueta leída para su posterior procesamiento:

```
while (eventType != XmlPullParser.END_DOCUMENT) {
    if (eventType == XmlPullParser.START_TAG) {
        String name = parser.getName();

        if (name.equals(Keys.errorCauseTag)) {
            return null;
        }

        if (name.equals(Keys.serialNumberTag)) {
            isSerialNumber = true;
        }

        if (name.equals(Keys.timestampTag)) {
```

Figura 39: Implementación del parser XML (II).

Cuando se llega a la etiqueta de cierre, se lanza el evento `END_TAG`. Se aprovecha este evento para cambiar el valor a un booleano que indique que se ha terminado de leer la etiqueta de cierre para así poder procesar el texto de las variables anteriormente guardadas.

```
} else if (eventType == XmlPullParser.END_TAG) {
    if (isSerialNumber) {
        serialNumber = text;
        isSerialNumber = false;
    }

    if (isTimestamp) {
        timestamp = text;
        isTimestamp = false;
    }

    if (isPhysicalQuality) {
```

Figura 40: Implementación del parser XML (III).

El evento `TEXT` indica que estamos posicionado en el contenido entre la etiqueta de apertura y la de cierre. Dicho contenido se almacena en otra variable para su posterior procesamiento.

```
} else if (eventType == XmlPullParser.TEXT) {
    text = parser.getText();
}
```

Figura 41: Implementación del parser XML (IV).

Cuando se hayan procesado todos los campos necesarios para crear el mensaje, creamos el mensaje y lo guardamos en un *array* de mensajes.

```
if (newMessage) {

    Message message = new Message(serialNumber, timestamp,
            physicalQuality, quantityValue, prefix, unitOfMeasure);
    messages.add(message);
    newMessage = false;
}
```

Figura 42: Implementación del parser XML (V).

La última iteración que realiza el bucle es pasar al evento siguiente:

```
eventType = parser.next();
```

Figura 43: Implementación del parser XML (VI).

En la siguiente figura se muestra un diagrama de actividad de un *parser XMLPull* genérico como los utilizados en la aplicación.

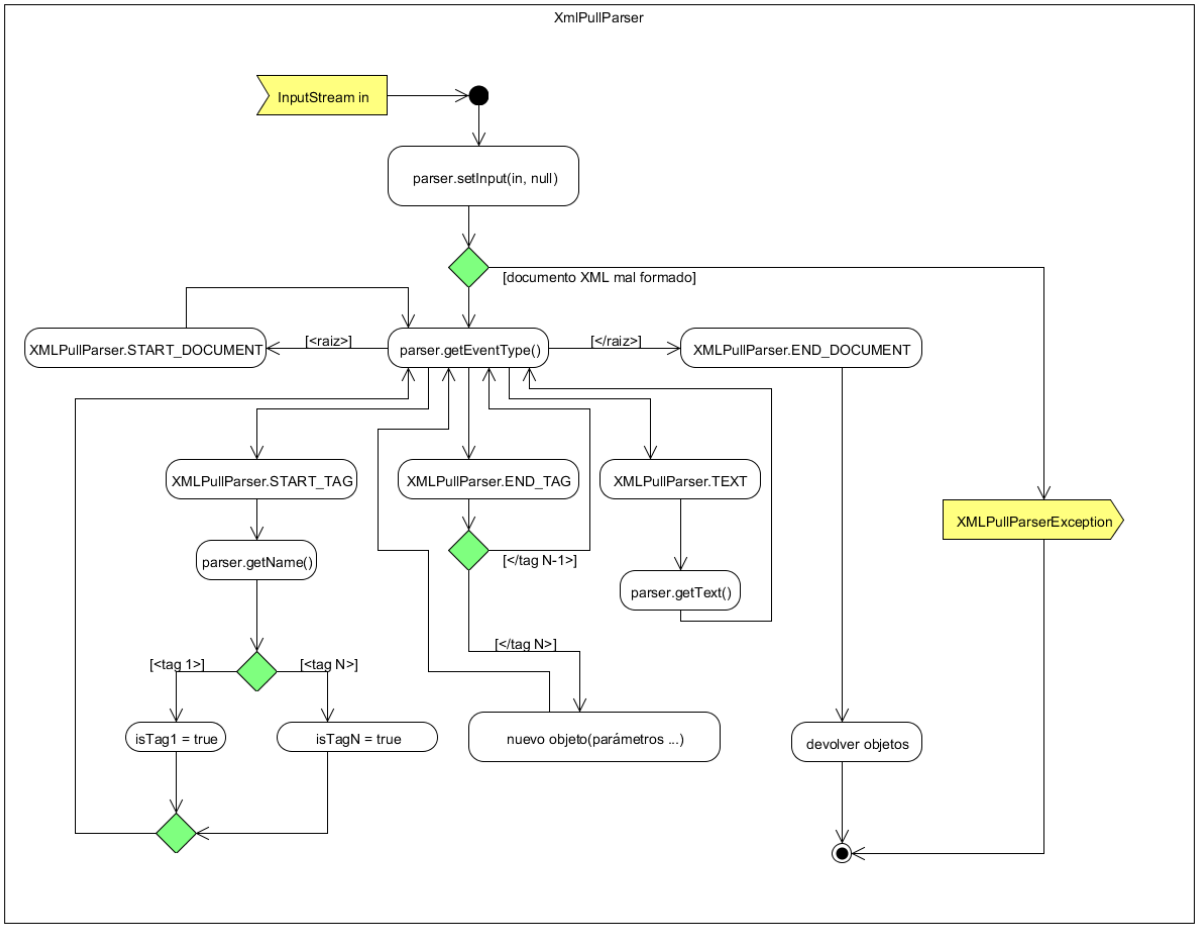

Figura 44: Diagrama de actividad de un parser XmlPull genérico.

5.4.4 Procesador SPARQL

Para realizar consultas semánticas, se hace uso de una API externa llamada Androjena (licencia Apache 2.0) [26], que es un *port* de Jena para Android. Se va a utilizar el procesador SPARQL de Jena para realizar las consultas semánticas.

Para poder utilizarla, hay que guardar las librerías de Androjena dentro de la carpeta /libs/ del protecto y después añadirlas al *Java Build Path* con las opción "Add JARs…".

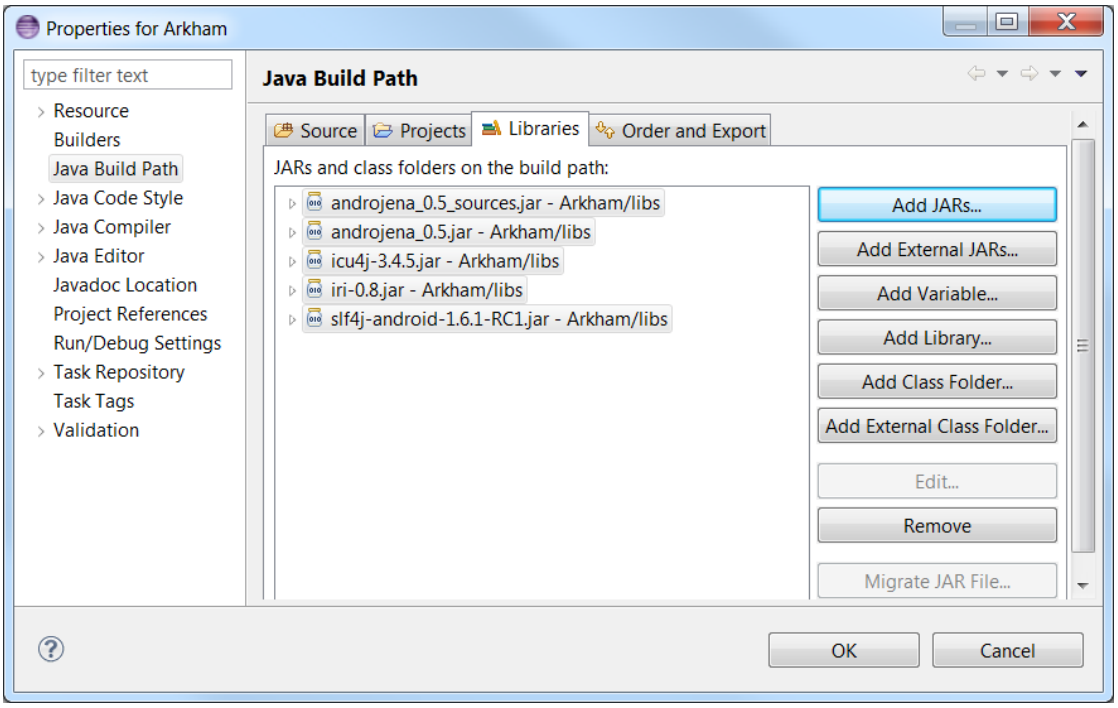

Figura 45: Importación de librerías al Java Build Path.

El siguiente paso es importar dentro de la clase los paquetes que vamos a utilizar:

```
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.Syntax;
```

Figura 46: Declaración en el código de la aplicación de las librerías de Jena.

- **Query**: Objeto que contiene la *query*.
- **QueryExecution**: Gestor de conexión con el *SPARQL endpoint*.
- **QueryExecutionFactory**: Factoría para instanciar un objeto de la clase *QueryExecution*.
- **QuerySolution**: Cada uno de los resultados de la consulta se almacena en un objeto de esta clase.
- **ResultSet**: Conjunto de objetos de la clase *QuerySolution*.
- **Syntax**: Define la sintaxis ARQ de las consultas.

Como se va a realizar una conexión a un servidor remoto, es necesario hacerlo dentro de una clase "AsyncTask", tal y como se ha detallado anteriormente.

```
/**
 * Implementation of AsyncTask to perform a semantic query to a SPARQL
 * endpoint.
 */
private class SparqlProcessor extends AsyncTask<String, Void, String> {
    ProgressDialog progressDialog = new ProgressDialog(SemanticQuery.this,
            ProgressDialog.STYLE_SPINNER);

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog.setMessage(getResources().getString(
                R.string.executing_query));
        progressDialog.show();
    }

    @Override
    protected String doInBackground(String... params) {
        return queryRemoteSparqlEndpoint();
    }

    @Override
    protected void onPostExecute(String result) {
        progressDialog.dismiss();

        if (result != null)
            textView.setText(result);
        else
            Toast.makeText(getBaseContext(), R.string.invalid_query,
                    Toast.LENGTH_SHORT).show();
    }
}
```

Figura 47: Implementación del procesador SPARQL mediante Jena (I).

En el siguiente código se detalla el proceso completo de realizar una consulta semántica utilizando la API de Androjena:

```
/**
 * Uses the SPARQL engine and report the results
 *
 * @return The number of resulting rows
 */
private String queryRemoteSparqlEndpoint() {
    // Setup a place to house results for output
    StringBuffer results = new StringBuffer();
```

Figura 48: Implementación del procesador SPARQL mediante Jena (II).

Primero, se configura la *query* y el *SPARQL endpoint*. El valor de la URI del *endpoint* se obtiene de las preferencias configuradas por el usuario:

```
// Set the query
String queryString = textField.getText().toString();

// Set the SPARQL endpoint URI
String sparqlEndpointUri = prefs.getString(Keys.endpointUriKey,
        Keys.blankValue);
```

Figura 49: Implementación del procesador SPARQL mediante Jena (III).

A continuación, se crea la instancia de la *query*, establecemos los parámetros de configuración; número máximo de resultados y el *offset*, que es la posición desde la que se recuperan los resultados (por defecto desde el primero):

```
// Create a Query instance
Query query = QueryFactory.create(queryString, Syntax.syntaxARQ);

// Limit the number of results returned
// Setting the limit is optional - default is unlimited
query.setLimit(resultsLimit);

// Set the starting record for results returned
// Setting the limit is optional - default is 1 (and it is 1-based)
query.setOffset(resultsOffset);

// Query uses an external SPARQL endpoint for processing
// This is the syntax for that type of query
QueryExecution qe = QueryExecutionFactory.sparqlService(
        sparqlEndpointUri, query);
```

Figura 50: Implementación del procesador SPARQL mediante Jena (IV).

Después, se ejecuta la consulta y se procesa con un bucle los resultados obtenidos:

```
try {
    // Execute the query and obtain results
    ResultSet resultSet = qe.execSelect();

    // Get the column names (the aliases supplied in the SELECT clause)
    List<String> columnNames = resultSet.getResultVars();

    // Iterate through all resulting rows
    while (resultSet.hasNext()) {
        // Get the next result row
        QuerySolution solution = resultSet.next();
```

Figura 51: Implementación del procesador SPARQL mediante Jena (V).

El siguiente paso es extraer la información de cada resultado y almacenarlo en un *buffer*:

```
// Iterate through the columns
for (String var : columnNames) {
    // Add the column label to the StringBuffer
    results.append(var + ": ");

    // Add the returned row/column data to the StringBuffer

    // Data value will be null if optional and not present
    if (solution.get(var) == null) {
        results.append("{null}");
        // Test whether the returned value is a literal value
    } else if (solution.get(var).isLiteral()) {
        results.append(solution.getLiteral(var).toString());
        // Otherwise the returned value is a URI
    } else {
        results.append(solution.getResource(var).getURI());
    }
    results.append('\n');
}
results.append("----------------\n");
```

Figura 52: Implementación del procesador SPARQL mediante Jena (VI).

A continuación, se cierra la conexión con el servidor para liberar los recursos reservados y se devuelven los resultados en una cadena de texto:

```
    .
} catch (Exception e) {
    e.printStackTrace();
} finally {
    // Important - free up resources used running the query
    qe.close();
}

// Return the results as a String
return results.toString();
```

Figura 53: Implementación del procesador SPARQL mediante Jena (VII).

5.4.5 Acceso al almacenamiento externo

Antes de acceder a la tarjeta SD para guardar o cargar los ficheros HTML o las plantillas con las consultas semánticas, se tienen que realizar dos comprobaciones:

- Poder escribir en la tarjeta SD.
- Poder leer la tarjeta SD.

Dichos permisos tiene que ser declarados previamente en el fichero "AndroidManifest.xml":

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Figura 54: Permisos de escritura y lectura en el almacenamiento externo.

El siguiente código los dos métodos utilizados para realizar dichas comprobaciones:

```
/**
 * Checks if external storage is available for read and write
 *
 * @return true if is available, false otherwise
 */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/**
 * Checks if external storage is available to at least read
 *
 * @return true if is available, false otherwise
 */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)
            || Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

Figura 55: Implementación de los métodos de comprobación del estado del almacenamiento externo.

5.4.6 Compartir ficheros

Para compartir los ficheros HTML que se han generado, primero se deben almacenar en la tarjeta SD con la opción “Save data to SD card”.

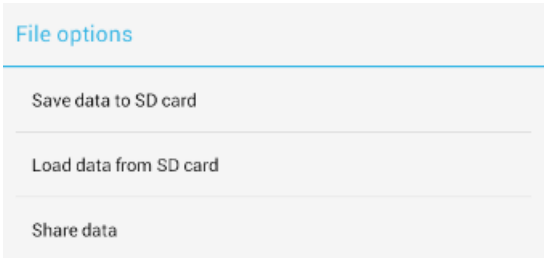

Figura 56: Diálogo con las opciones de fichero.

Cuando se selecciona la opción “Share data” se invoca el siguiente método:

```
/**
 * Called when the "Share data" option is clicked
 */
public void shareData() {
    if (isExternalStorageReadable()) {
        File sdcard = Environment.getExternalStorageDirectory();
        File[] files = sdcard.listFiles();
        final ArrayList<String> filesArray = new ArrayList<String>();

        for (int i = 0; i < files.length; i++) {
            String name = files[i].getName();
            if (name.endsWith(fileExt))
                filesArray.add(name);
        }

        if (filesArray.size() == 0)
            Toast.makeText(getBaseContext(), R.string.files_not_found,
                    Toast.LENGTH_SHORT).show();
```

Figura 57: Implementación del método para compartir ficheros por correo electrónico (I).

A continuación, se muestra al usuario un dialogo con los ficheros disponibles:

Figura 58: Diálogo con los ficheros HTML disponibles para ser compartidos.

Una vez que el usuario selecciona el fichero a compartir, este se carga desde la tarjeta SD:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle(R.string.available_files)
        .setItems(
                filesArray.toArray(new CharSequence[filesArray
                        .size()]),
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog,
                            int which) {
                        String fileName = filesArray.get(which);
                        File sdcard = Environment
                                .getExternalStorageDirectory();
                        File file = new File(sdcard, fileName);
                        Uri outputFileUri = Uri.fromFile(file);
```

Figura 59: Implementación del método para compartir ficheros por correo electrónico (II).

Dicho fichero se adjunta al mensaje de correo electrónico que va a ser enviado, junto con los demás campos configurados por el usuario (remitente, CC, asunto y mensaje). Para ello, se instancia un objeto de la clase "Intent", que es la encargada en Android para este tipo de tarea. El tipo de mensaje rfc822 hace referencia al código MIME [27] de los mensajes correos electrónicos.

```
Intent shareIntent = new Intent();
shareIntent
        .setAction(Intent.ACTION_SEND);
shareIntent
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);
shareIntent.putExtra(
        Intent.EXTRA_STREAM,
        outputFileUri);
shareIntent.putExtra(
        Intent.EXTRA_EMAIL,
        address);
shareIntent.putExtra(
        Intent.EXTRA_CC, cc);
shareIntent.putExtra(
        Intent.EXTRA_SUBJECT,
        subject);
shareIntent.putExtra(
        Intent.EXTRA_TEXT,
        message);
shareIntent
        .setType("message/rfc822");

startActivity(Intent
        .createChooser(
                shareIntent,
                title));
```

Figura 60: Implementación del método para compartir ficheros por correo electrónico (III).

5.5 Funcionamiento de la aplicación

5.5.1 Pantalla principal

En la Figura 60 se ilustra la pantalla principal de la aplicación, que es la que se muestra cuando se inicia la aplicación y desde la que se accede a las principales funcionalidades:

- **Registry Information**: Obtiene la información de los servicios y dispositivos registrados.
- **Measures**: Obtiene los datos de medidas de los dispositivos (corriente, temperatura, etc).
- **Semantic Query**: Proporciona una interfaz para realizar consultas semánticas a al *SPARQL endpoint* configurado.
- **Settings**: Permite al usuario configurar los diferentes parámetros de la aplicación. Necesita permisos de administrador.
- **About**: Información sobre el autor y la versión de la aplicación.
- **Admin mode**: Acceso al modo administrador. Se solicita una contraseña al usuario. Permite habilitar la opción “Settings”.

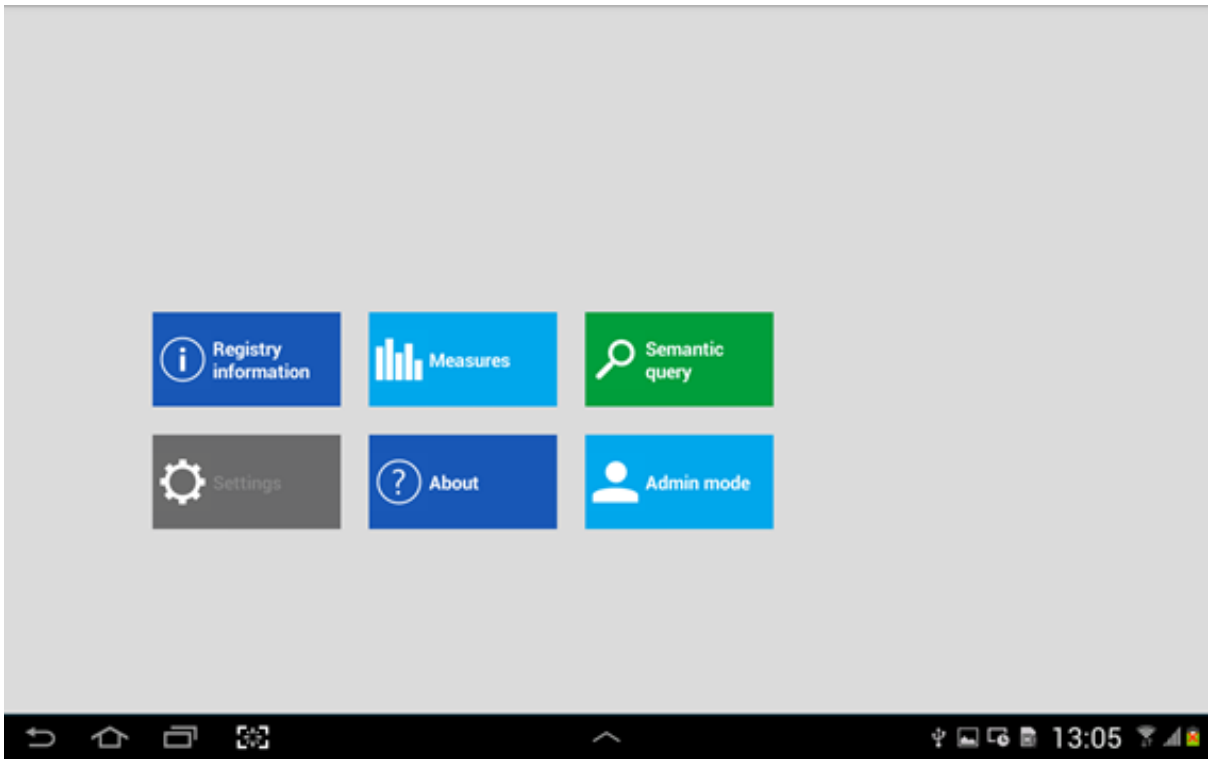

Figura 61: Pantalla principal de la aplicación.

5.5.2 Información de registro

La siguiente pantalla se muestra cuando el usuario selecciona la opción "Registry Information". En ella se observa la información de registro de los diferentes dispositivos registrados en la *Smart Grid*.

- **Device Id**: Identificador único del dispositivo.
- **Model**: Modelo del dispositivo.
- **Manufacturer**: Fabricante del dispositivo.
- **Serial Number**: Número de serie del dispositivo.
- **Location**: Localización del dispositivo. Se proporciona un enlace a *Google Maps* [25] donde se visualiza en un mapa la ubicación del dispositivo.

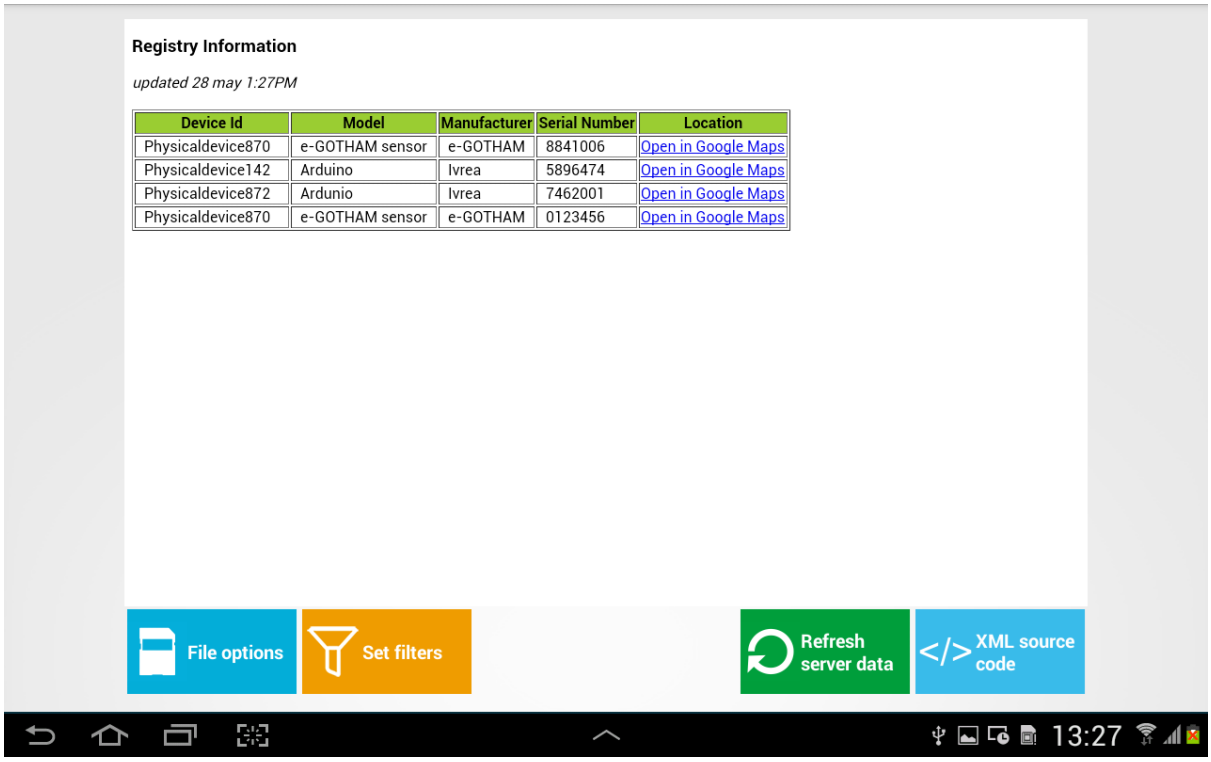

Figura 62: Tabla con la información de registro.

En la parte inferior de la pantalla existen 4 utilidades:

- **File options**: Opciones de fichero. Muestra un dialogo con las operaciones guardar, cargar o compartir el fichero HTML desde la tarjeta SD.
- **Set filters**: Configurar filtros. Permite establecer filtros por identificador de dispositivo, modelo o fabricante.
- **Refresh server data**: Actualizar datos del servidor. Solicita al servidor de nuevo el fichero XML con la información de registro. En la pantalla se muestra la fecha y la hora a la que se obtuvo el fichero.
- **XML source code**: Cambia la vista entre el fichero HTML y el fichero XML obtenido en el servidor. Pensado para temas de depuración.

La siguiente captura muestra la ubicación de un dispositivo en *Google Maps*:

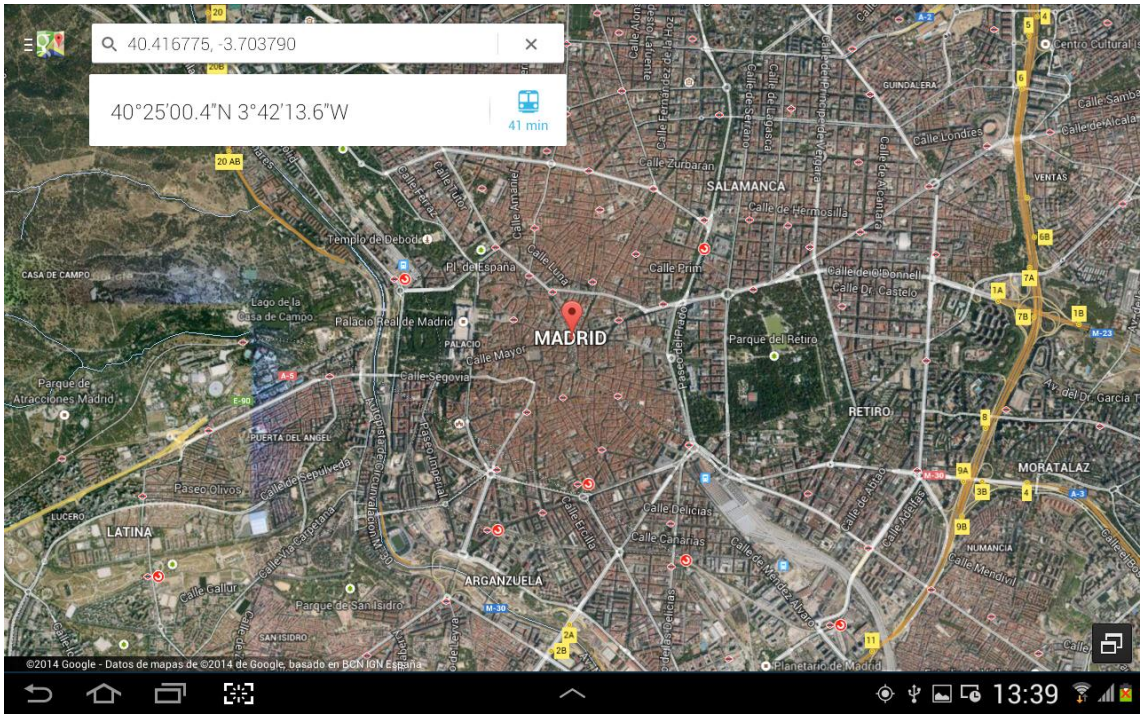

Figura 63: Ubicación de un dispositivo en Google Maps.

A continuación se muestra un dialogo con las posibles opciones de fichero:

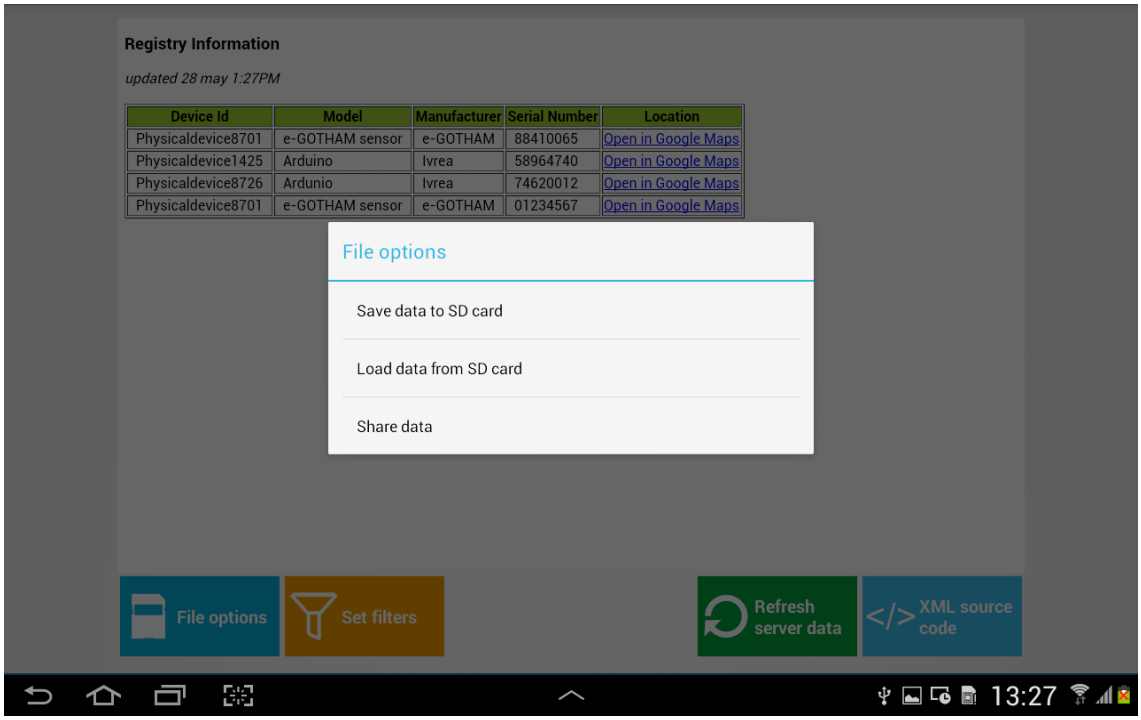

Figura 64: Opciones de fichero.

La siguiente captura muestra un mensaje de correo electrónico con el fichero con el fichero HTML adjunto:

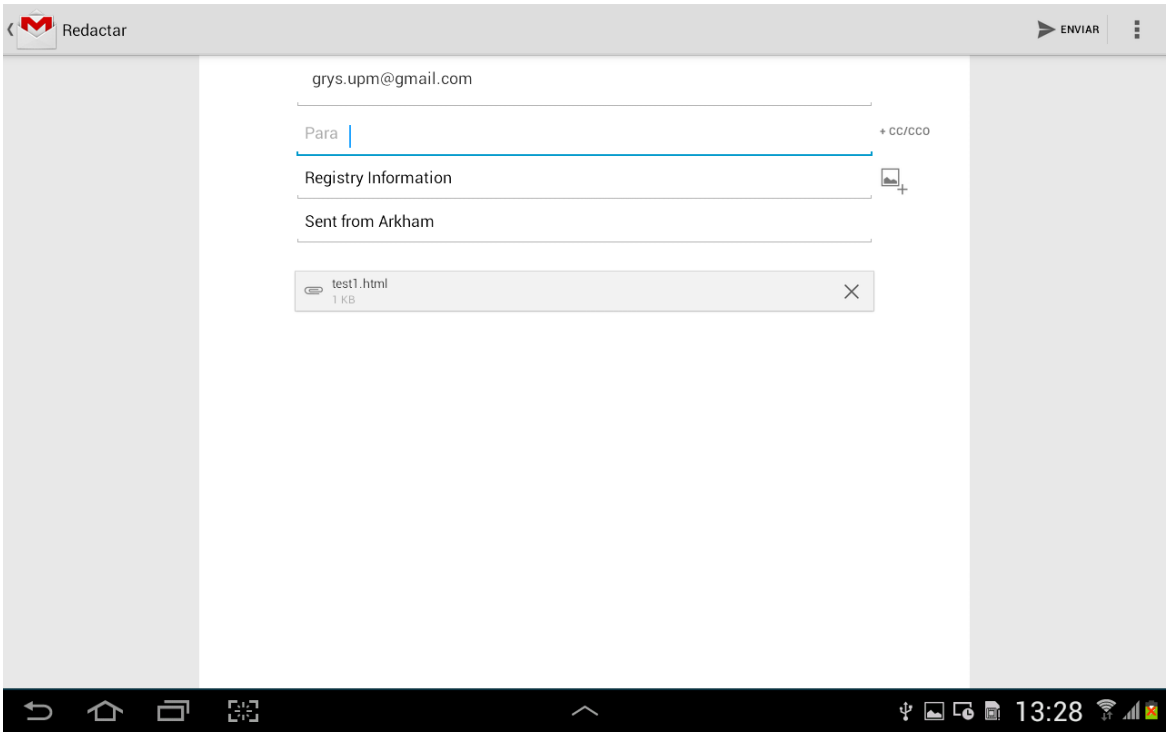

Figura 65: Envío del fichero HTML por correo electrónico.

Diálogo con la configuración de los filtros:

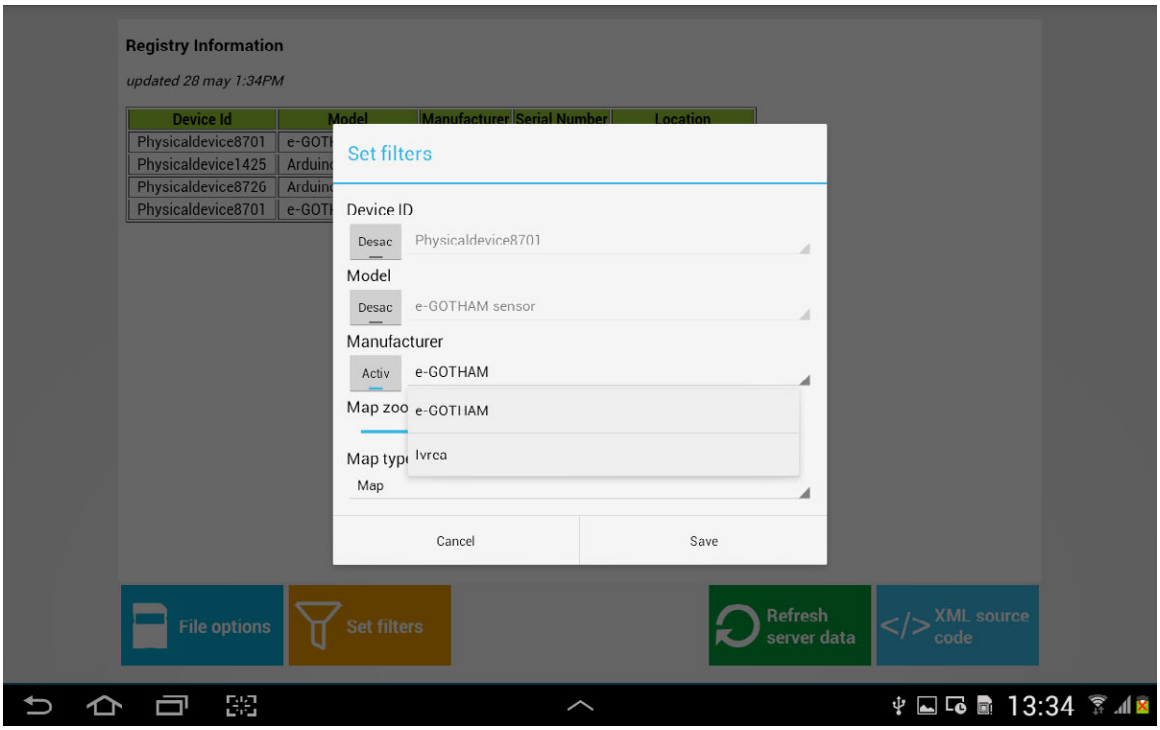

Figura 66: Configuración de filtros.

5.5.3 Medidas

La siguiente captura muestra una tabla con los datos medidos por un dispositivo. En este caso, se trata de una medida de voltaje y el dispositivo viene identificado por su número de serie. Cuando se pulsa la opción "Refresh server data", se actualizará el valor de la medida, obteniendo así el dato en tiempo real (Considerando despreciable el tiempo que se tarda el dispositivo en publicar la medida al servidor y lo que se tarda en descargar y procesar el fichero XML).

- **Serial number**: Número de serie del dispositivo.
- **Timestamp**: Sello de tiempo. Indica el instante en el que se realizó la medida.
- **Physical quality**: Magnitud física medida. En el siguiente ejemplo, corriente eléctrica.
- **Quantity value**: Valor medido.
- **Prefix**: Prefijo del Sistema Internacional de Unidades (SI) que indica el factor por el que multiplicar.
- **Unit of measure**: Unidad de medida.

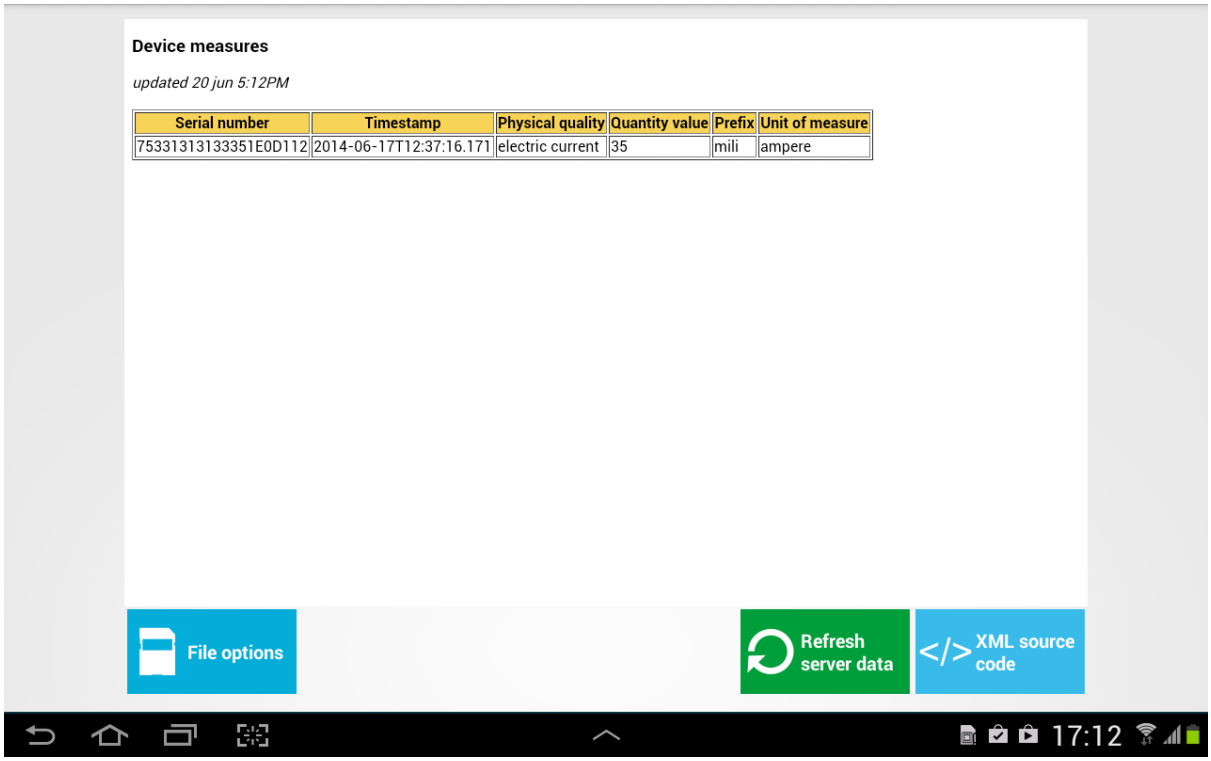

Figura 67: Tabla con las medidas en tiempo real.

5.5.4 Consultas semánticas

A continuación se muestra la interfaz desde donde se realizan las consultas semánticas. En la parte superior existe un formulario donde escribir la consulta (“Query text”). Para mayor comodidad, las consultas pueden cargarse desde un fichero de texto que esté almacenado en la tarjeta SD, utilizando la utilidad “Template options”. El siguiente paso es pulsar el botón “Run query”. Los resultados aparecen en el visor de texto inferior (“Results”). En el ejemplo siguiente, se modela mediante una consulta SPARQL la siguiente pregunta: ¿De los países que no linden con ningún mar, cuales tienen una población estimada mayor de 15.000.000?

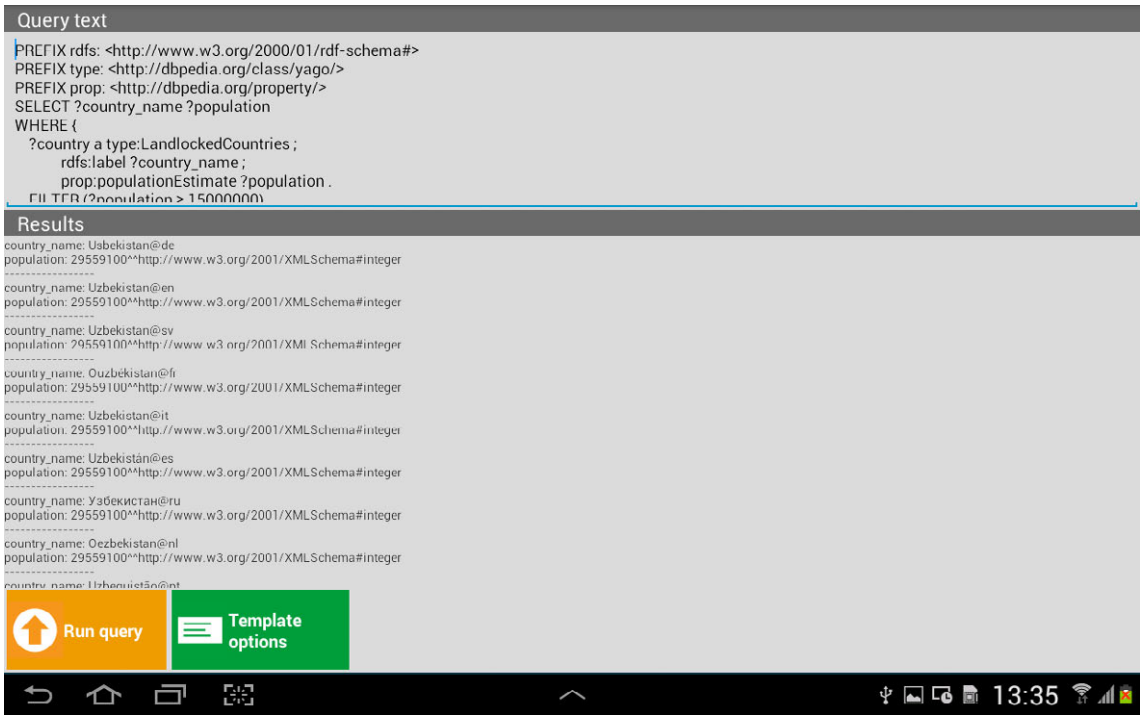

Figura 68: Ejemplo de consulta semántica.

5.5.5 Ajustes y modo administrador

La siguiente captura muestra la pantalla de ajustes de la aplicación, a la que se accede con permisos de administrador:

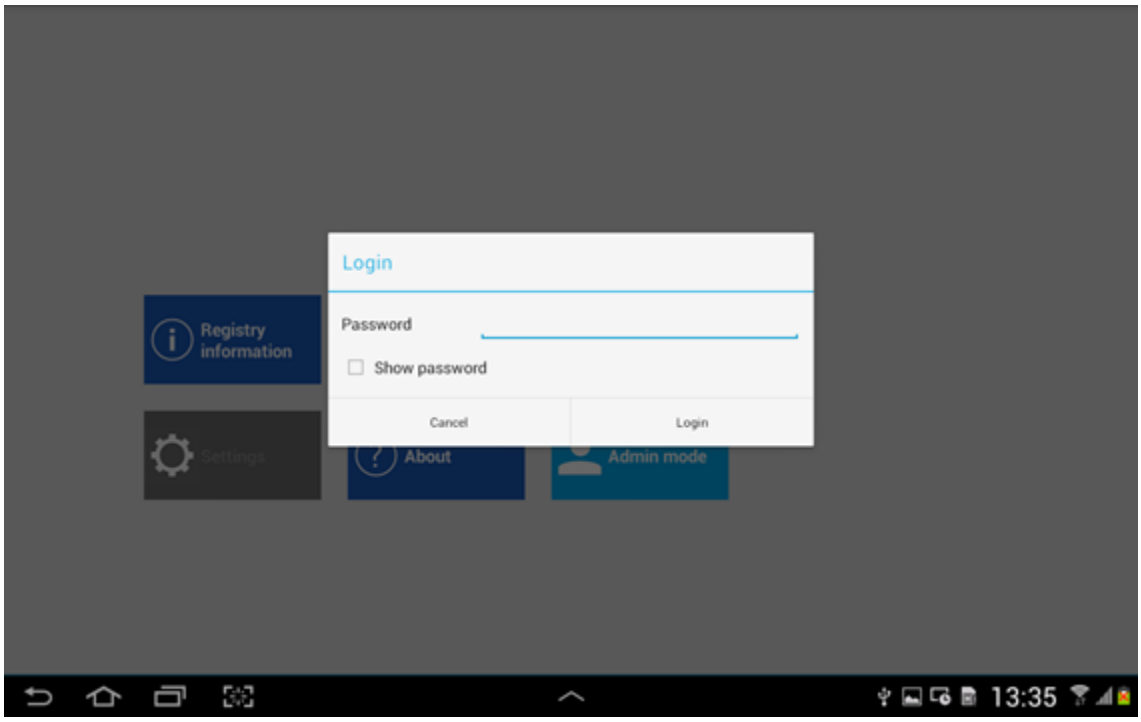

Figura 69: Diálogo de acceso al nodo administrador.

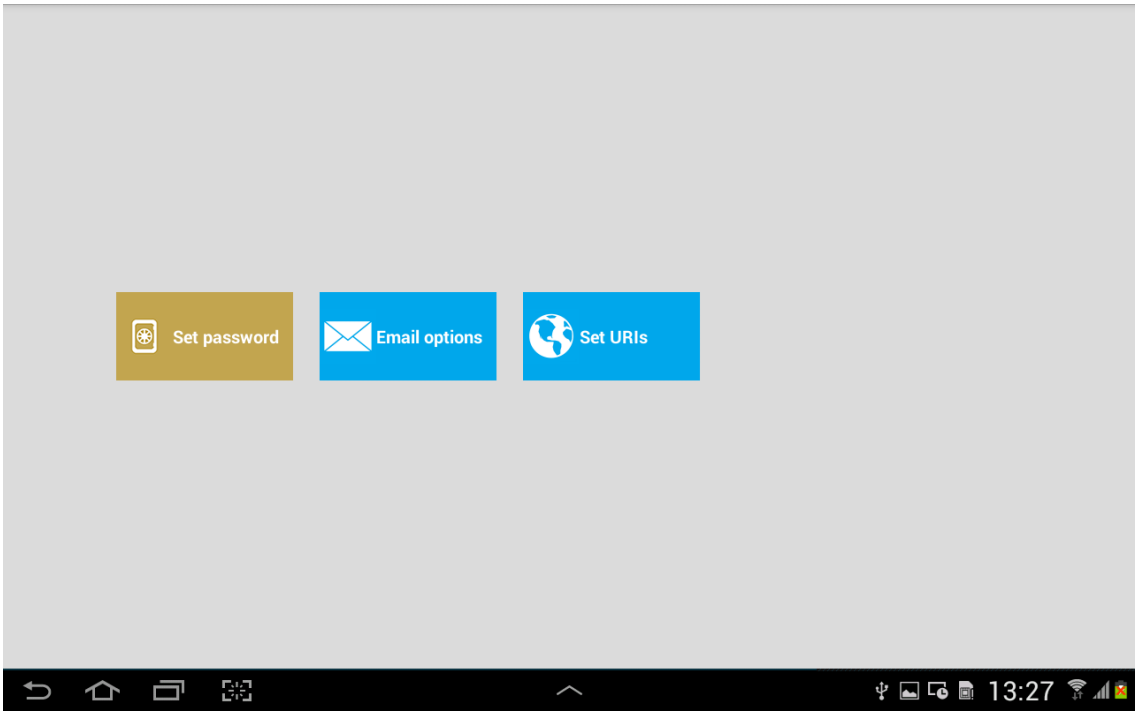

Figura 70: Ajustes de la aplicación.

6 Conclusiones

Este Proyecto Fin de Grado ha pretendido demostrar que las arquitecturas basadas en servicios suponen una buena opción a tener en cuenta en el desarrollo de proyectos de *Smart Grids*, ya que proporcionan las herramientas adecuadas para satisfacer las necesidades de gestión de una red inteligente.

El *middleware* será una de las piezas angulares dentro de este nuevo concepto de red inteligente, mediante la integración de bases de datos semánticas, sistemas distribuidos y redes de sensores.

Las tareas de estandarización para la interoperabilidad de los distintos sistemas, tendrán una trascendencia fundamental en la transición hacia la red inteligente. Se requerirán nuevos estándares y la ampliación o adaptación de los mismos.

Será necesario impulsar el I+D+i en el ámbito de las telecomunicaciones. Las universidades y empresas privadas jugaran un papel clave en esta cuestión, creando nuevas alianzas beneficiosas para ambas partes.

La parte de implementación ha consistido en la creación de una herramienta abierta para la creación de interfaces de gestión de información en *Smart Grids*, de la que no se tiene constancia de que exista nada similar actualmente en el mercado. Con ligeras modificaciones en el código de los *parsers*, puede adaptarse a cualquier plataforma SOA o similares, lo que la convierte en una herramienta muy útil para diferentes proyectos de *Smart Grids*.

La herramienta desarrollada constituye una buena base para, a partir de la cual, desarrollar, como trabajos futuros, nuevos *parsers* para adaptarlos a futuros servicios RESTful y plataformas. También se podrán implementar nuevas características como el *login* del administrador contra un servidor de autentificación, ampliar las opciones del menú de configuración o extender la funcionalidad del procesador SPARQL para que devuelva los resultados en otros formatos (como por ejemplo, XML).

Por último, agradecer al GRyS permitirme utilizar la infraestructura del proyecto e-Gotham para las pruebas de la aplicación, así como el acceso a todos los recursos *hardware* y *software* necesarios para el desarrollo de la misma.

7 Referencias

[1] «F.A.Q.,» Smart Grids European Technology Platform, [En línea]. Disponible: http://www.smartgrids.eu/FAQ. [Último acceso: Marzo 2014].

[2] «¿Qué son las Smart Grids?,» Red Eléctrica Española (REE), [En línea]. Disponible: http://www.ree.es/es/red21/redes-inteligentes/que-son-las-smartgrid. [Último acceso: Marzo 2014].

[3] «Visión sobre Smart Grids,» VSE Smart Energy, [En línea]. Disponible:: http://vse-smartenergy.com/wp/?portfolio=¿que-es-smart-grid. [Último acceso: Marzo 2014].

[4] CITCEA – UPC (Universitat Politècnica de Catalunya), «Smart Grids: Sectores y actividades clave,» Funseam, [En línea]. Disponible: www.funseam.com/informes-funseam?download=24. [Último acceso: Mayo 2014].

[5] «Objetivos del proyecto e-Gotham,» e-Gotham, [En línea]. Disponible: http://www.e-gotham.eu/index.php/e-gothamproj/objectives. [Último acceso: Marzo 2014].

[6] «Conceptos del proyecto,» I3RES, [En línea]. Disponible: http://www.i3res.eu/v1/index.php/project2/concept2. [Último acceso: Marzo 2014].

[7] «About Arrowhead,» Arrowhead, [En línea]. Disponible: http://www.arrowhead.eu/about/. [Último acceso: Marzo 2014].

[10] «SOA Architecture,» Oracle, [En línea]. Disponible: http://docs.oracle.com/cd/E13171_01/alsb/docs26/concepts/introduction.html. [Último acceso: Marzo 2014].

[11] T. Erl, SOA Principles of Service Design, New Jersey: Prentice Hall, 2007.

[12] «The OSGi Architecture,» OSGi Alliance, [En línea]. Disponible: http://www.osgi.org/Technology/WhatIsOSGi. [Último acceso: Marzo 2014].

[13] «AR - Jini Architecture Specification,» Apache Software Foundation, [En línea]. Disponible: http://river.apache.org/doc/specs/html/jini-spec.html. [Último acceso: Marzo 2014].

[15] «UPnP Device Architecture,» UPnP Forum, [En línea]. Disponible: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf. [Último acceso: Marzo 2014].

[16] «Simple Service Discovery Protocol (SSDP),» W3C, [En línea]. Disponible: http://www.w3.org/TR/discovery-api. [Último acceso: Marzo 2014].

[17] «UDDI 3.0 Specification,» OASIS, [En línea]. Disponible: http://uddi.org/pubs/uddi-v3.0.2-20041019.htm. [Último acceso: Marzo 2014].

[18] T. Gruber, «Toward principles for the design of ontologies used for knowledge sharing,» *Journal of Human-Computer Studies,* vol. 43, nº 4-5, pp. 907-928, 1995.

[19] «Lenguaje de Ontologías Web (OWL),» W3C, [En línea]. Disponible: http://www.w3.org/2007/09/OWL-Overview-es.html. [Último acceso: Abril 2014].

[20] «Web Service Modeling Language (WSML),» W3C, [En línea]. Disponible: http://www.w3.org/Submission/WSML. [Último acceso: Abril 2014].

[21] «Web Service Modeling Ontology (WSMO),» W3C, [En línea]. Disponible: http://www.w3.org/Submission/WSMO. [Último acceso: Abril 2014].

[22] A. W. Mcmorran, «An Introduction to IEC 61970-301 & 61968-11: The Common Information Model,» University of Strathclyde, Glasgow, 2007.

[24] «SPARQL Query Language for RDF,» W3C, [En línea]. Disponible: http://www.w3.org/TR/rdf-sparql-query. [Último acceso: Abril 2014].

[25] «API Android Developer. AsyncTask Class,» Google, [En línea]. Disponible: http://developer.android.com/reference/android/os/AsyncTask.html. [Último acceso: Mayo 2014].

[26] «Androjena. Jena Android porting,» [En línea]. Disponible: https://code.google.com/p/androjena. [Último acceso: Mayo 2014].

[27] «The Message Content-Type,» W3C, [En línea]. Disponible: http://www.w3.org/Protocols/rfc1341/7_3_Message.html. [Último acceso: Mayo 2014].